

PROBABILISTIC ONTOLOGY:
REPRESENTATION AND MODELING METHODOLOGY

by

Rommel Novaes Carvalho

A Dissertation

Submitted to the

Graduate Faculty

of

George Mason University

in Partial Fulfillment of

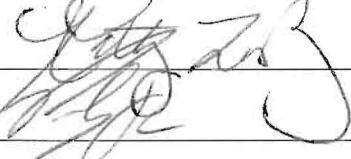
The Requirements for the Degree

of

Doctor of Philosophy

Systems Engineering and Operations Research

Committee:



Dr. Kathryn Laskey, Dissertation Director



Dr. Paulo Costa, Committee Member



Dr. Kuo-Chu Chang, Committee Member



Dr. David Schum, Committee Member



Dr. Larry Kerschberg, Committee Member



Dr. Fabio Cozman, Committee Member



Dr. Ariela Sofer, Department Chair



Dr. Lloyd J. Griffiths, Dean, Volgenau School
of Engineering

Date: 28 June 2011

Summer Semester 2011
George Mason University
Fairfax, VA

Probabilistic Ontology:
Representation and Modeling Methodology

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at George Mason University

By

Rommel Novaes Carvalho
Master of Science
University of Brasília - UnB, 2008
Bachelor of Science
University of Brasília - UnB, 2003

Director: Dr. Kathryn Laskey, Professor
Department of Systems Engineering and Operations Research

Summer Semester 2011
George Mason University
Fairfax, VA

Copyright © 2011 by Rommel Novaes Carvalho
All Rights Reserved

Dedication

I dedicate this work to my wife who supported me with love and wisdom during this marvelous and arduous experience, to my parents and sisters who have showed me my whole life, through example, that hard work always pays off, and to my future children who even before being born have inspired me to be the best example possible, both as a father and as a person.

Acknowledgments

I have been blessed with so much help from friends and colleagues that I could easily make this section the longest in my whole dissertation. However, in consideration to you, the reader, I will try to be concise without forgetting to thank all the people that helped me, somehow, fulfill my scholar's dream, which is manifested in this dissertation.

First, I would like to thank the Brazilian Office of the Comptroller General (CGU) for believing in my work and giving me the time and money to work full-time on my PhD. I would like to particularly thank my bosses José Geraldo Loureiro Rodrigues, Oswaldo Iglesias de Azeredo, and Tatiana Zolhof Panisset for supporting my proposal when I requested the time necessary (and the extension) to come here to advance science in order to provide better tools for improving public transparency and fighting corruption. On the same note, I would like to thank the Office of Naval Research (ONR) and the Air Force Office of Scientific Research (AFOSR) for hiring me as a Graduate Research Assistant (GRA) during my PhD, under Contract #N00173-09-C-4008 and Grant #FA9550-08-1-0425, respectively. This experience as a GRA has given me the opportunity to put my ideas into practice and allowed me to participate and present papers at several conferences. I am also grateful for the fellowships I received during my PhD: the ISWC travel fellowship I received from the U.S. National Science Foundation (NSF) that partially funded my trip to China for the ISWC conference, and the Volgenau School of Engineering (VSE) Academic Fellowship from George Mason University that helped me with expenses during Spring 2011.

As previously described, many people have contributed to the conclusion and even initiation of this PhD. I would like to start by thanking the people who helped me write the almost 100 pages long report I submitted to CGU asking for permission to work on my PhD. In particular, I would like to thank Prof. Dr. Marcelo Ladeira for not only reviewing the document many times, but also helping me write many parts of the document. Moreover, I would like to thank Gustavo Gomes and Heleno Borges for having the patience to read this long report and giving me important insights on how to make it *much* better several times. Last, but not least, I would like to thank the folks from the Corruption Prevention and Strategic Information Secretariat (SPCI) for helping me with the proof of concept use case that showed how my work with probabilistic ontologies could be applied for fighting corruption. In special, I would like to thank Henrique Rocha who believed in the idea and Mário Vinícius Claussen Spinelli, the expert who trained me on techniques for fraud detection and prevention and evaluated the proof of concept model I created.

Once I arrived in the USA and actually started working on my dissertation, I started getting help from folks all over the world, given the diversity on the student body at George Mason University. A group that I am extremely grateful to have been part of is the Krypton group. Every Friday we would have somebody presenting their work and receiving feedback from not only Professors, but also students and even folks working in Industry. More importantly, this was a venue that allowed me to dry run many presentations and to validate many of my ideas, which eventually became my PhD. I would like to thank the group as a

whole for having the patience to hear about procurement fraud detection about a thousand times, at times, even more than one Friday in a row! I am truly grateful. In particular, I would like to thank Andy Powell, Bill Bunting, Charles Twardy, Mark Locher, Todd Martin, Wei Sun and Young Park for their extensive feedback.

Additional thanks goes to all the people involved in the PRobabilistic OntoloGies for Net-centric OperationS (PROGNOS) project. Some of them have already been mentioned here, so they will not be repeated. The ones not yet mentioned are Dr. Paulo Costa, Dr. Kathryn Laskey, Dr. KC Chang, Aditya Mugali, Richard Harbelin, and Michael Lehocky. The last two were the subject matter experts (SME) on one of the use cases in this dissertation. A special thanks goes to Rick Harbelin for helping me out a lot on this use case in numerous meetings and e-mail exchanges.

The professors deserve a separate thanks given the amount of work and patience they had with me. First, I would like to thank my tutor and friend Dr. Paulo Costa, who has helped me even before I came to the USA. He was the co-advisor on my Masters thesis, where I implemented his PhD work, PR-OWL. Without him I am sure this PhD project would have not even started. Besides being a good teacher, Paulo has been a great friend. I lost count of the number of barbecues, I mean meetings, and beers, I mean discussions, we had both at his and my place. Not only that, but his family has also been great company to both me and my dear wife, be it in Fairfax, Scotland, or Chicago!

Dr. Kathryn Laskey has been tough as a scholar and kind as a mother. I remember going to her office after just a few months in the PhD program to ask if my work was living up to her expectation. I remember telling her that I was afraid I was not being productive enough, since there were a lot of things I did not understand and that I was taking too long to read and learn all those papers I had been getting from her and Dr. Chang. Then she looked at me really serious and simply said to me: “Rommel, that is why you are doing your PhD! If you knew it all you would not be here! Just give yourself some time.” Dr. Laskey was also a baseline for me to see how much I was learning and evolving in my research. I remember in our first meetings that I would come up with a great idea on how to fix my algorithm and she would look at it and “destroy” it in less than 30 seconds! I had no idea how she did that. How can she come up with a counter-example so fast? The only thing I could do was to take her counter-examples, go home, and think about them for days before I could come back to continue our discussion. But then, as weeks turned into months and months into years, I started to realize that I did not have to take her counter-examples home anymore. I actually started to counter her counter-examples on the fly! Wow, was that exciting or what? I would get anxious to go home and tell my wife the news that I was able to counter-argue something that Dr. Laskey said! How cool was I? Things kept getting better and better as my ideas actually matured enough that she would even agree with them without having some kind of counter-example. By the end of my PhD I even had some cases where I was able to have new reasonable ideas during a meeting with Dr. Laskey and even actively participate in discussions with her providing counter-examples of my own! I still have a lot to learn, but she sure was a good baseline to track my progress during these 3 years. Just one last example that she still has a lot to teach me is how she transformed my horrible and cumbersome 3-page abstract into less than one and a half well-written pages. The amazing thing to notice was that she did not leave anything out, put it in a much more understandable way, and in half the size! But that is not all. Besides being a great teacher, Dr. Laskey helped me on many occasions with her overhead funding

to support my trips to conferences. Without her help I would not have been able to go to ISWC/URSW 2010 (in China) nor to Fusion 2011 (in Chicago). As I said, tough as a scholar and kind as a mother! Thanks, Dr. Laskey!

Dr. Chang has been a great Professor and teacher. First of all, I would like to thank him for the fact that he agreed to let me work on his project, even without knowing me, while I was waiting for the PROGNOS project to start. Besides that, I've learned a lot from the few months we have worked together in this project. It is amazing how he can make extremely difficult math papers so easy to understand with just 10 minutes of explanation and a few sketches on a blank page.

However, these were not the only professors who helped me. I would like to thank all the professors I worked with at the Volgenau School of Engineering from George Mason University. I would like to highlight the professors who participated in my committee (besides the ones I have already thanked), which includes Dr. David Schum, Dr. Larry Kerschberg, and Dr. Fabio Cozman, who is from the University of São Paulo and kindly accepted my invitation even without the funding to bring him personally to participate in my comprehensive exam, pre-defense, and defense.

One group that I do not get tired of thanking is the Group of Artificial Intelligence (*Grupo de Inteligencia Artificial - GIA*) from the University of Brasília, Brazil. I lost count of the number of times they have helped me. I would like to thank, in particular, Dr. Marcelo Ladeira who is responsible for the group and who I have been working with since 2001. I would also like to thank Laécio Santos for helping out with implementation of a few algorithms we worked on for MEBN after a few meetings with Dr. Laskey through Skype during my first year of my PhD studies. Finally, I would like to thank Shou Matsumoto. Shou has helped me not only with the implementation of my PhD work for his Masters degree, but also by testing out my ideas and giving me important feedback and insights on how to improve my PhD work. For that I am truly grateful.

Furthermore, I would like to thank all my friends and family who have helped me directly or indirectly. Even PhD students deserve a break and although I have not been able to visit my home country, Brazil, for three years, I have received countless and unforgettable visits from friends and family. It is true that I was not able to go to Brazil because there was always somebody visiting us during our break, but my wife and I have enjoyed each and every one of the visits! Thank you all for making this intense work more enjoyable.

Finally, I would like to thank again my wife, Silnara Batista Carvalho, who has helped me so many times that it is impossible to keep track. She would encourage me to relax when I was working too much, but would also demand that I had to work when I was too relaxed. Thank you so much for taking care of me, for being my partner, my lover, and for making me so happy no matter the circumstances. I have seen our love grow and with it our maturity as a couple. Last but not least, I would like to thank one more time my parents, Francisco de Moura Carvalho Neto and Mírian Novaes Carvalho, and sisters, Patrícia Novaes Carvalho and Tatiana Novaes Carvalho, who have supported me in every way possible, be it financially or emotionally. Thank you so much for believing in me and for giving me all the support a child and brother could ever ask for!

Table of Contents

	Page
List of Tables	x
List of Figures	xi
Abstract	xvi
1 Introduction	1
1.1 Problem Statement	2
1.1.1 Lack of mapping to OWL	6
1.1.2 Lack of support for OWL types	8
1.2 Research Contributions and Structure of this Dissertation	9
2 Different Approaches To Knowledge Modeling	12
2.1 UML and ER	13
2.2 Knowledge Representation and Reasoning	17
2.3 Ontology and the Semantic Web	20
2.3.1 The Advantages of Ontology and the Semantic Web	21
2.3.2 The Beginning of OWL	22
2.3.3 The Web Ontology Language (OWL)	25
3 Representing Uncertainty	28
3.1 Multi-Entity Bayesian Network (MEBN)	29
3.2 Probabilistic Web Ontology Language (PR-OWL)	30
3.3 Related Work	35
3.3.1 First-Order Probabilistic Languages (FOPL)	35
3.3.2 Probabilistic Languages for the SW	39
4 A Formal Definition for Probabilistic Ontology - PR-OWL 2	41
4.1 Why map PR-OWL Random Variables to OWL Properties?	42
4.2 The bridge joining OWL and PR-OWL	46
4.3 Extending PR-OWL to Use OWL's Types	51
4.4 Defining a Random Variable in PR-OWL 2	53
4.4.1 Mutually Exclusive and Collectively Exhaustive Outcomes	56
4.4.2 Avoiding OWL Full	57

4.4.3	Built-in Random Variables	58
4.4.4	Defining Arguments for Random Variables	62
4.4.5	Defining Distributions for Random Variables	63
4.4.6	Examples of Random Variables	65
4.5	Entity Hierarchy and Polymorphism	72
4.6	Type Uncertainty	76
4.7	Defining Nodes in PR-OWL 2	78
4.7.1	Defining Domain-Specific Knowledge	80
4.7.2	Defining Findings	82
4.7.3	MEBN Expressions	84
4.7.4	Examples of Nodes	86
4.8	Types of Uncertainty Reasoning for the Semantic Web	101
5	Uncertainty Modeling Process for Semantic Technologies (UMP-ST)	104
5.1	Probabilistic Ontology for Procurement Fraud Detection and Prevention in Brazil	109
5.1.1	Requirements	111
5.1.2	Analysis & Design	115
5.1.3	Implementation	121
5.1.4	Test	129
5.2	Probabilistic Ontology for Maritime Domain Awareness	138
5.2.1	First Iteration	139
5.2.2	Second Iteration	149
5.2.3	Third Iteration	162
5.2.4	Testing the Final MDA PO	175
6	Conclusion	182
6.1	Future Work	185
A	PR-OWL 2 Abstract Syntax and Semantics	189
A.1	Random Variables	190
A.2	MEBN Main Elements	197
A.3	MEBN Expressions	228
B	Use Cases Implementation Details	246
B.1	Probabilistic Ontology for Procurement Fraud Detection and Prevention in Brazil	246
B.2	Probabilistic Ontology for Maritime Domain Awareness	266
B.2.1	Fist Iteration	267
B.2.2	Second Iteration	280

B.2.3	Third Iteration	295
Bibliography		310

List of Tables

Table	Page
4.1 Table representing the distribution for the random variable <code>isRelated(person1, person2)</code>	67
5.1 Requirements Traceability Matrix for the requirements of the fraud detection model	116
5.2 Requirements Traceability Matrix for the rules of the fraud detection model	119
5.3 Requirements Traceability Matrix for the MFrags of the fraud detection model	127
5.4 A simple method for identifying entities, attributes, and relationships	165
5.5 Grouping for entities, attributes, and relations in third iteration	168
5.6 Number of TP, FP, TN, and FN	179
5.7 Percentage of TP, FP, TN, and FN	180
A.1 Table representing the distribution for the random variable <code>isRelated(person1, person2)</code>	221
A.2 Table representing the distribution for the domain resident node <code>isRelated(person1, person2)</code>	222

List of Figures

Figure	Page
1.1 A regular system for storing information about public procurements	3
1.2 An automated system for identifying and preventing frauds in public procurements	4
1.3 A class diagram for the procurement domain	7
1.4 Front of an Enterprise MFragment	8
1.5 Boolean individual defined in PR-OWL	9
2.1 A class diagram for the procurement domain	14
2.2 A class diagram with instances of classes from the procurement domain	15
2.3 An extended entity-relationship diagram for the procurement domain	16
2.4 Retrieving information through the SW for the procurement domain	23
3.1 PR-OWL simple model	33
3.2 PR-OWL detailed model	34
3.3 A taxonomy of first-order probabilistic languages (reproduced with permission from Milch and Russell [91])	37
4.1 OWL ontology for the public procurement domain	42
4.2 Front of an Enterprise MFragment	43
4.3 Using OWL triples for probabilistic reasoning	45
4.4 Unknown mapping between PR-OWL 1 RVs and OWL properties	45
4.5 PR-OWL 1 lack of mapping from RVs to OWL properties	46
4.6 PR-OWL 1 lack of mapping from arguments to OWL properties	47
4.7 The bridge joining OWL and PR-OWL	48
4.8 Example of binary RVs mapping to OWL properties for both predicate and function	50
4.9 Boolean individual defined in PR-OWL	52
4.10 The different types of entities defined in PR-OWL	53
4.11 The OWL restrictions of the <code>RandomVariable</code> class	55
4.12 Graph with the main concepts for defining random variables	59
4.13 The OWL restrictions of the <code>BooleanRandomVariable</code> class	59

4.14	The OWL restrictions of the classes <code>LogicalOperator</code> and <code>Quantifier</code>	60
4.15	The OWL restrictions of the <code>MappingArgument</code> class	62
4.16	The OWL restrictions of the <code>ProbabilityDistribution</code> class	64
4.17	Graph with the main concepts and their relations for defining probabilistic distributions	65
4.18	Graph of the main MEBN elements and their relations	79
4.19	Graph with the main elements necessary to define a domain-specific MEBN fragment	81
4.20	Graph with the main concepts and their relations for defining findings in an MEBN theory	82
4.21	Bayesian network showing the pattern for representing findings in MEBN	83
4.22	Graph with main concepts and their relations necessary for defining arguments and MEBN expressions	85
4.23	Nodes for representing that two people are more likely to be related if they live at the same address	86
4.24	A common way to define restrictions like symmetry in BNs	87
4.25	Posterior probabilities for symmetrical properties showing how the constraint works, but unfortunately it double counts evidence	88
4.26	Defining restrictions like symmetry in BNs using order on arguments	89
4.27	Posterior probabilities for symmetrical properties using order on arguments, which does not cause double counting of evidence	90
4.28	Nodes for representing the finding that Bill has annual income of 75,000.00	97
5.1	Uncertainty Reasoning Process for ST (URP-ST)	105
5.2	Uncertainty Modeling Process for Semantic Technologies (UMP-ST)	107
5.3	Probabilistic Ontology Modeling Cycle (POMC) - Requirements (Goals), Analysis & Design (Entities, Rules, and Group), Implementation (Mapping and LPD), and Test (Evaluation)	108
5.4	Procurement fraud detection overview	111
5.5	Entities, their attributes, and relations for the procurement model	118
5.6	Entities for the procurement domain	122
5.7	Creating a RV in PR-OWL 2 plugin from its OWL property by drag-and-drop	124
5.8	Probabilistic ontology for fraud detection and prevention in public procurements	126
5.9	LPD for node <code>isSuspiciousProcurement(procurement)</code>	128

5.10	Results of unit testing for the <code>JudgmentHistory_MFrag</code>	131
5.11	Part of the SSBN generated for the first scenario	134
5.12	Part of the SSBN generated for the second scenario	136
5.13	Part of the SSBN generated for the third scenario	138
5.14	Entities, their attributes, and relations for the MDA model after the first iteration	141
5.15	MTheory created in first iteration	144
5.16	SSBN generated for scenario 1	145
5.17	SSBN generated for scenario 2	146
5.18	SSBN generated for scenario 3	147
5.19	SSBN generated for scenario 4	148
5.20	SSBN generated for scenario 5	149
5.21	Normal and suspicious behavior of merchant and fishing ships	150
5.22	Entities, their attributes, and relations for the MDA model after the second iteration	154
5.23	MTheory created in second iteration	156
5.24	SSBN generated for the first scenario	159
5.25	SSBN generated for the second scenario	160
5.26	SSBN generated for the third scenario	162
5.27	MFrags changed/added in third iteration	170
5.28	SSBN generated for scenario 1	172
5.29	SSBN generated for scenario 2	173
5.30	SSBN generated for scenario 3	175
5.31	Part of the SSBN generated for “sure” positive of a possible exchange illicit cargo using merchant ship	177
5.32	Simulation editor	180
6.1	Example of how to use EPF to tailor the UMP-ST process	188
A.1	The hierarchy of PR-OWL 2 classes	189
A.2	The OWL restrictions of the <code>RandomVariable</code> class	190
A.3	The hierarchy of the <code>RandomVariable</code> class	191
A.4	Graph with the main concepts for defining random variables	192
A.5	The OWL restrictions of the <code>BooleanRandomVariable</code> class	194
A.6	The OWL restrictions of the classes <code>LogicalOperator</code> and <code>Quantifier</code> . .	195
A.7	The hierarchy of the main classes that represent MEBN elements	197
A.8	The OWL restrictions of the <code>MTheory</code> class	198

A.9	Graph of the main MEBN elements and their relations	199
A.10	The OWL restrictions of the MFrag class	200
A.11	The OWL restrictions of the DomainMFrag class	201
A.12	Graph with the main elements necessary to define a domain-specific MEBN fragment	202
A.13	The OWL restrictions of the FindingMFrag class	203
A.14	Graph with the main concepts and their relations for defining findings in an MEBN theory	204
A.15	Bayesian network showing the pattern for representing findings in MEBN .	205
A.16	The OWL restrictions of the Node class	206
A.17	The OWL restrictions of the ResidentNode class	207
A.18	The OWL restrictions of the DomainResidentNode class	208
A.19	The OWL restrictions of the FindingResidentNode class	210
A.20	The OWL restrictions of the ContextNode class	211
A.21	The OWL restrictions of the InputNode class	212
A.22	The OWL restrictions of the FindingInputNode class	213
A.23	The OWL restrictions of the GenerativeInputNode class	215
A.24	The OWL restrictions of the ProbabilityDistribution class	216
A.25	Graph with the main concepts and their relations for defining probabilistic distributions	217
A.26	The OWL restrictions of the DeclarativeDistribution class	218
A.27	The OWL restrictions of the PR-OWLTable class	220
A.28	The OWL restrictions of the classes ProbabilityAssignment and ConditioningState	224
A.29	The hierarchy of the main classes for representing arguments and MEBN expressions	229
A.30	Graph with main concepts and their relations necessary for defining arguments and MEBN expressions	230
A.31	The OWL restrictions of the MExpression class	231
A.32	The OWL restrictions of the SimpleMExpression class	231
A.33	The OWL restrictions of the BooleanMExpression class	233
A.34	The OWL restrictions of the Argument class	234
A.35	The OWL restrictions of the ConstantArgument class	235
A.36	The OWL restrictions of the OrdinaryVariableArgument class	237
A.37	The OWL restrictions of the ExemplarArgument class	238

A.38	The OWL restrictions of the <code>MExpressionArgument</code> class	240
A.39	The OWL restrictions of the <code>MappingArgument</code> class	242
A.40	The OWL restrictions of the <code>OrdinaryVariable</code> class	243
A.41	The OWL restrictions of the <code>Exemplar</code> class	245
B.1	MFrag Personal Information	249
B.2	MFrag Procurement Information	251
B.3	MFrag Enterprise Information	253
B.4	MFrag Front of Enterprise	254
B.5	MFrag Exists Front in Enterprise	256
B.6	MFrag Related Participant Enterprises	257
B.7	MFrag Member Related to Participant	258
B.8	MFrag Competition Compromised	259
B.9	MFrag Owns Suspended Enterprise	261
B.10	MFrag Judgment History	262
B.11	MFrag Related to Previous Participants	263
B.12	MFrag Suspicious Committee	264
B.13	MFrag Suspicious Procurement	265
B.14	MFrag for identifying the ship of interest	268
B.15	MFrags for identifying a terrorist crew member	269
B.16	MFrags for identifying the ship with unusual route	273
B.17	MFrags for identifying the ship with evasive behavior	276
B.19	Ship Characteristics MFrag	282
B.18	MTheory created in second iteration	283
B.20	Terrorist Plan MFrag	284
B.21	Bomb Port Plan MFrag	285
B.22	Exchange Illicit Cargo Plan MFrag	286
B.23	Ship of Interest MFrag	287
B.24	Meeting MFrag	288
B.25	Unusual Route MFrag	289
B.26	Evasive Behavior MFrag	291
B.27	Aggressive Behavior MFrag	292
B.28	Erratic Behavior MFrag	293
B.29	MTheory created in third iteration	299

Abstract

PROBABILISTIC ONTOLOGY:
REPRESENTATION AND MODELING METHODOLOGY

Rommel Novaes Carvalho, PhD

George Mason University, 2011

Dissertation Director: Dr. Kathryn Laskey

The past few years have witnessed an increasingly mature body of research on the Semantic Web (SW), with new standards being developed and more complex problems being addressed. As complexity increases in SW applications, so does the need for principled means to cope with uncertainty in SW applications. Several approaches addressing uncertainty representation and reasoning in the SW have emerged. Among these is Probabilistic Web Ontology Language (PR-OWL), which provides Web Ontology Language (OWL) constructs for representing Multi-Entity Bayesian Network (MEBN) theories. However, there are several important ways in which the initial version PR-OWL 1 fails to achieve full compatibility with OWL. Furthermore, although there is an emerging literature on ontology engineering, little guidance is available on the construction of probabilistic ontologies.

This research proposes a new syntax and semantics, defined as PR-OWL 2, which improves compatibility between PR-OWL and OWL in two important respects. First, PR-OWL 2 follows the approach suggested by Poole *et al.* to formalizing the association between random variables from probabilistic theories with the individuals, classes and properties from ontological languages such as OWL. Second, PR-OWL 2 allows values of random variables to range over OWL datatypes.

To address the lack of support for probabilistic ontology engineering, this research describes a new methodology for modeling probabilistic ontologies called the Uncertainty Modeling Process for Semantic Technologies (UMP-ST). To better explain the methodology and to verify that it can be applied to different scenarios, this dissertation presents step-by-step constructions of two different probabilistic ontologies. One is used for identifying frauds in public procurements in Brazil and the other is used for identifying terrorist threats in the maritime domain. Both use cases demonstrate the advantages of PR-OWL 2 over its predecessor.

Chapter 1: Introduction

The same assumptions that were essential in the document web are still applied for the Semantic Web (SW). They are radical notions of information sharing, which include [7]:

(i) the Anyone can say Anything about Any topic (AAA) slogan; (ii) the open world assumption, i.e. there might exist more information out there that we are not aware of, and (iii) nonunique naming, meaning that different people can assign different names to the same concept. However, the Semantic Web differs from its predecessors in the sense that it intends to provide an environment not only for allowing information sharing but also for making it possible to have the effect of knowledge synergy. Nevertheless, this can lead to a chaotic scenario with disagreements and conflicts.

I call an environment characterized by the above assumptions a Radical Information Sharing (RIS) environment. The challenge facing SW architects is therefore to avoid the natural chaos to which RIS environments are prone, and move to a state characterized by information sharing, cooperation and collaboration. According to Allemang and Hendler [7], one solution to this challenge lies in modeling. A model is a simplified abstraction of some real world phenomenon, which, amongst other things, allows the organizing of information for community use. Modeling is the process of constructing such a simplified abstraction. Modeling supports information sharing in three ways: it provides a means for human communication, it provides a way for explaining conclusions, and it provides the managing of different viewpoints.

There is an immense variety of modeling processes, which, in turn, are supported by different modeling languages. One of especial interest in this research is the Web Ontology Language (OWL) [89, 104] which was intended to enable the achievement of the Semantic Web full potential. According to [89] OWL “is intended to be used when the information

contained in documents needs to be processed by applications, as opposed to situations where the content only needs to be presented to humans. OWL can be used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. This representation of terms and their interrelationships is called an ontology.”

One of the first definitions of ontology in the context of the Semantic Web was given by Thomas Gruber [54].

An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an Ontology is a systematic account of Existence. For Artificial Intelligence (AI) systems, what “exists” is that which can be represented. A conceptualization is an abstract, simplified view of the world that we wish to represent for some purpose. Every knowledge base, knowledge-based system, or knowledge-level agent is committed to some conceptualization, explicitly or implicitly.

Appreciation is growing within the Semantic Web community of the need to represent and reason with uncertainty. In recognition of this need, the World Wide Web Consortium (W3C) created the Uncertainty Reasoning for the World Wide Web Incubator Group (URW3-XG) in 2007 to identify requirements for reasoning with and representing uncertain information in the World Wide Web. The URW3-XG concluded that standardized representations were needed to express uncertainty in Web-based information [82]. A candidate representation for uncertainty reasoning in the Semantic Web is Probabilistic OWL (PR-OWL) [27], an OWL upper ontology for representing probabilistic ontologies based on Multi-Entity Bayesian Networks (MEBN) [78].

1.1 Problem Statement

Imagine an application where we try to identify and prevent frauds in public procurements. Figure 1.1 shows a common approach to the problem. Most of the information associated to public procurements are available in physical documents (Public Notices - Data). When a

procurement is audited, experts analyze those documents in search of information that will provide evidence that the procurement process was done as described by law or that the procurement is suspicious because it did not follow the expected procedures (Information Gathering). Finally, that information is then stored in a database where it can be accessed later (DB - Information). The problem the experts usually complain about is that they spend a lot of time feeding all this information to the database, but the system does not provide any useful knowledge, regarding fraudulent activities, based on the information provided. *I.e.*, the experts still have to search through all that data in order to find evidence of suspicious activities for every procurement.



Figure 1.1: A regular system for storing information about public procurements.

The problem is that there are many more procurements than experts capable of auditing them. For example, the Brazilian Transparency Portal¹ alone has over 1 billion items of information stored in its database. The database covers over 5 trillion US Dollars in Government expenditures, including procurements. This clearly shows that the experts are experiencing an overload of information that makes it impossible to analyze every single procurement.

A reasonable solution to this problem of information overload is to change the focus from

¹This portal (<http://www.portaldatransparencia.gov.br>) is an initiative of the Brazilian Office of the Comptroller General (CGU) that started in 2004. Its main objective is to make the expenditures of public money more transparent by allowing any citizen to follow where the money is being spent in order to help supervise and audit the Brazilian Government



Figure 1.2: An automated system for identifying and preventing frauds in public procurements.

data driven to knowledge driven, as shown in Figure 1.2. In other words, we need some kind of automated solution that has the experts' knowledge represented in some model (Design - UnBBayes²). Then machines can use this model to infer new knowledge, for instance, if a procurement is suspicious of fraudulent activities, for all procurements and using all the information available in a reasonable time (Inference - Knowledge). Then this knowledge can be filtered so that only the procurements that show a probability higher than a threshold, *e.g.* 20%, are automatically forwarded to the responsible department along with the inferences about potential fraud and the supporting evidence (Report for Decision Makers). Probabilistic Ontologies (POs) can be used to represent the experts' knowledge in order to allow inferences like the one just described.

However, as a possible language for representing POs, PR-OWL still presents some

²The text in parenthesis refers to the block in the figure with the same label.

problems. As stated by Costa [27], a major design goal for PR-OWL was to attain compatibility with OWL. However, to date this goal has been only partially achieved, mostly due to a couple of key issues not fully addressed in the original work. First, there is no mapping in PR-OWL to properties of OWL. Second, although PR-OWL has the concept of meta-entities, which allows the definition of complex types, it lacks compatibility with existing types already present in OWL.

These problems have been noticed and stated in the literature. According to Predoiu and Stuckenschmidt [113]:

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

Furthermore, one major problem is that probabilistic ontologies are complex and hard to model. It is challenging enough to design models that use only logic or only uncertainty; combining the two poses an even greater challenge. In fact, in the past few years I have received a number of e-mails from researchers all around the world asking for information and/or literature on how to build probabilistic ontologies. The problem is that there is no published methodology specifically focused on probabilistic ontology engineering. Furthermore, probabilistic ontology engineering is a very challenging problem – something that requires a significant learning curve. Therefore, this is a major issue.

Although there is now substantial literature about what PR-OWL is [27, 29, 31], how to implement it [23, 20, 19, 26], and where it can be used [30, 32, 33, 77, 79, 80], little has been written about how to model a probabilistic ontology.

This lack of methodology is not only associated with PR-OWL. Other languages that use probabilistic methods for representing uncertainty on the SW have been advancing in areas like inference [12, 122], learning [36, 86], and applications [14, 120, 87], but little has been written on how to create such models from scratch. Example of such languages are

OntoBayes [136], BayesOWL [37], and probabilistic extensions of SHIF(**D**) and SHOIN(**D**) [85]. Moreover, even first-order probabilistic languages like Markov Logic Network (MLN) [40], which is also used for uncertainty reasoning on the SW [13,41], have the same limitation on methodologies for creating models.

In the following subsections I will give one example for each of the current limitations of PR-OWL related to its compatibility with OWL.

1.1.1 Lack of mapping to OWL

Suppose we have a well defined ontology described in OWL about the public procurement domain. In such a domain, we would have a well defined semantics for concepts like procurement, winner of a procurement, members of a committee responsible for a procurement, etc. Figure 1.3 shows a light-weight ontology for this domain represented in Unified Modeling Language (UML). For more details about UML, see Section 2.1.

Now, imagine we want to define some uncertain relations about this domain, *e.g.*, it is common to identify a front for an enterprise by looking at his/her income and the value of a procurement the enterprise he/she represents won. For instance, if the enterprise won a procurement of millions of dollars, but the responsible person for this enterprise makes less than 10 thousand dollars a year, it is likely that this person is a front. Figure 1.4 shows this probabilistic relation defined using PR-OWL in an open-source tool for probabilistic reasoning, UnBBayes.

As it is expected, we have to make sure some conditions hold to be able to make assertions about this probabilistic relationship. One of them is that the person we are trying to determine as a possible front has to be responsible for the enterprise we are analyzing. These conditions are represented in Figure 1.4 by the pentagon nodes (in green). The probabilistic relation mentioned is represented by the trapezoid and rounded rectangle nodes (in gray and yellow, respectively). The pentagon nodes are context nodes, the rounded rectangle nodes are resident nodes, which represent a random variable (RV) and its local probability distribution (LPD), and the trapezoid nodes are input nodes, which have their

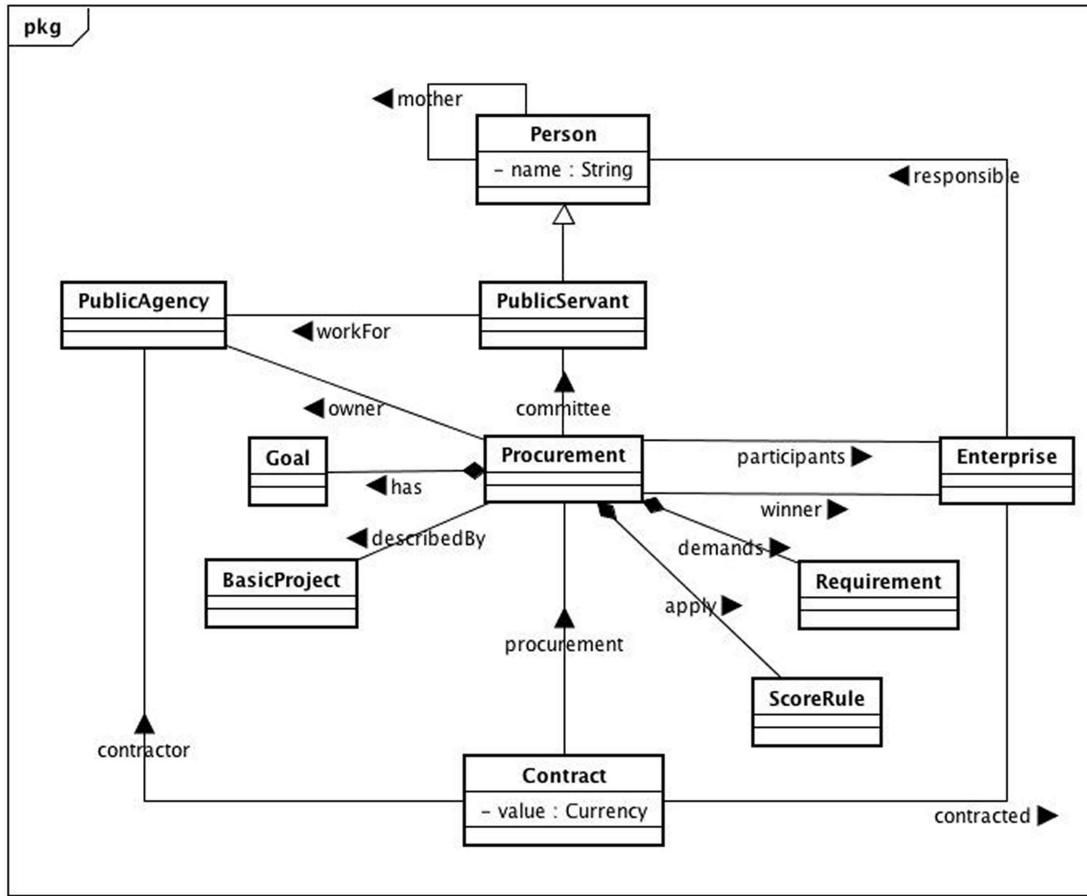


Figure 1.3: A class diagram for the procurement domain.

LPD defined somewhere else. For more details about MEBN, see Section 3.1.

It is natural to assume that the data we have about this domain will be associated to the ontological markups defined in OWL. In other words, our database will have instances of persons and enterprises associated to their semantic meaning defined in OWL.

Having access to this information should be trivial once the domain ontology has been defined and permission has been granted to retrieve data from the database. However, this can only be achieved by developing a link between PR-OWL random variables (RVs) and the concepts defined in OWL. The problem is that PR-OWL has no formal way to define a link between RVs and OWL concepts.

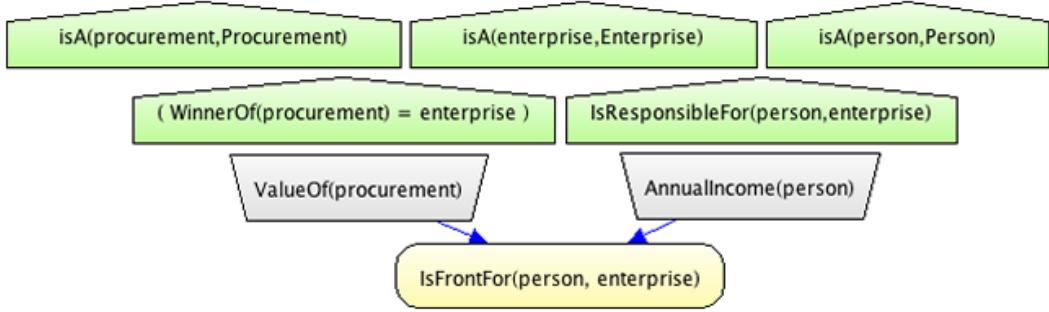


Figure 1.4: Front of an Enterprise MFragment.

With this simple example, it is clear that every probabilistic extension made to a concept has to keep a reference to its semantic definition, otherwise this definition will only be a standard random variable without the knowledge of the semantics defined on the deterministic ontology. That is, in order to achieve full compatibility with OWL, modifications to PR-OWL that guarantee the preservation of OWL's semantics are required.

1.1.2 Lack of support for OWL types

One of the main concerns when developing OWL [66] was to keep the same semantics of its predecessors, RDF and XML, which meant reusing all the concepts already defined in those languages, including primitive types, such as string, Boolean, decimal, etc. On the other hand, PR-OWL does not make use of the primitive types used in OWL. For instance, PR-OWL defines *Boolean* as an individual of the class *MetaEntity*, as shown in Figure 4.9, but does not keep any relation to the Boolean type used in OWL.

If we wanted to define a continuous random variable for the annual income of a person in PR-OWL, we would need to define the real numbers, even though they are already defined in OWL. Moreover, concepts that use this primitive type in OWL would not be understood in PR-OWL, in other words, they lack compatibility as far as primitive types are concerned.

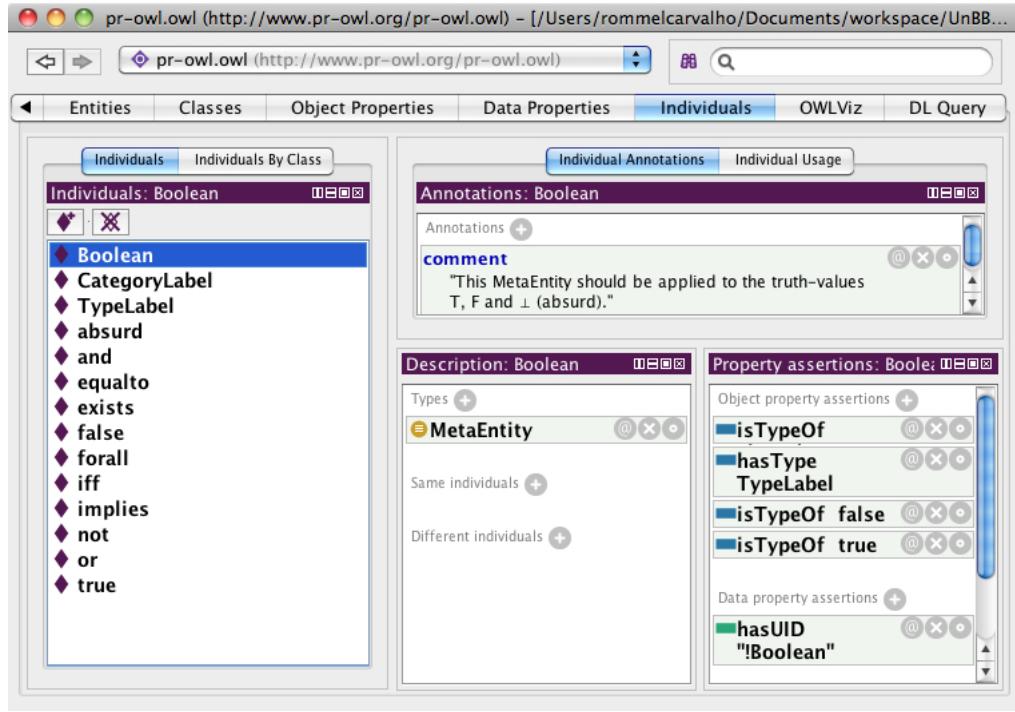


Figure 1.5: Boolean individual defined in PR-OWL.

1.2 Research Contributions and Structure of this Dissertation

Before I can describe how the limitations explained were addressed, Chapter 2 introduces different approaches to modeling and how they differ from the perspective of achieving the Semantic Web full potential. In it, it becomes evident the important role of ontological languages, more specifically, of OWL. Then, Chapter 3 presents various extensions to First-Order Logic (FOL) and ontology languages for allowing the representation of uncertainty. In it, I explain the importance of being able to reason with uncertain information. Moreover, besides explaining the basics of MEBN and PR-OWL 1, I also present related work for representing uncertainty in the SW.

The problem of compatibility is discussed in Chapter 4. It describes the changes made to PR-OWL by PR-OWL 2 to achieve full compatibility with OWL. The difficulty of this

kind of integration between probabilistic theories like PR-OWL and ontology languages like OWL has been discussed in the literature. Poole *et. al.* [110] emphasizes that it is not clear how to match the formalization of random variables from probabilistic theories with the concepts of individuals, classes and properties from current ontological languages like OWL. However, Poole *et. al.* [110] says that “We can reconcile these views by having properties of individuals correspond to random variables.” This is the approach taken in this work to integrate MEBN logic and the OWL language. Details of the syntax and semantics of PR-OWL 2 are presented in Appendix A.

To address the lack of support in probabilistic ontology engineering, Chapter 5 describes the new methodology for modeling probabilistic ontologies for semantic technologies. To better explain how the methodology works and to verify it can be applied to different scenarios, I describe step-by-step the construction of two different models from scratch. One model is used for identifying frauds in public procurements in Brazil³ and the other is used for identifying terrorist threats on the maritime domain⁴. In both cases, the advantages are highlighted of PR-OWL 2 over PR-OWL 1.

Finally, Chapter 6 summarizes the work presented in this dissertation and presents future directions for PR-OWL as well as for probabilistic ontology representation and modeling for the Semantic Web as a whole.

The contributions of this work can be summarized as:

1. A formal and extended definition of PR-OWL including the connection between a statement in PR-OWL and a statement in OWL.
2. PR-OWL 2 syntax - an upper-ontology that captures the formal definition described above.
3. PR-OWL 2 semantics - a clear specification of those aspects of the mappings from

³This use case has been developed with the support of the Brazilian Office of the Comptroller General (CGU), which has been providing the human resources and the information necessary to conduct this research since 2008.

⁴This use case was developed as part of the PROGNOS project [32], which has been partially supported by the Office of Naval Research (ONR), under Contract#: N00173-09-C-4008.

PR-OWL to OWL for which OWL has no formal semantics.

4. A proof of concept tool which allows the use of PR-OWL 2 to model probabilistic ontologies developed as a plugin for UnBBayes.
5. Methodology for modeling PO, the Uncertainty Modeling Process for the Semantic Web (UMP-SW).
6. Two use cases which use PR-OWL 2 and shows its benefits when compared to the current version of PR-OWL.

Chapter 2: Different Approaches To Knowledge Modeling

Before analyzing different approaches to knowledge modeling, we will define some terms that will be used from now on:

Classes represent concepts, which are understood in a broad sense. For instance, in the procurement domain, the `Goal` class represents a specific objective that needs to be achieved.

Instances are used to represent elements or individuals. For instance, `build a bridge` and `buy 500 chairs` might be specific individuals of the class `Goal`.

Relations represent a link between classes. Although it is possible to represent a relation between many classes at once, the most common is a binary relation, where the first argument is known as the domain of the relation, and the second argument is the range. For instance, classes are usually organized in taxonomies through which inheritance mechanisms can be applied. The binary relation `subclassOf` is used to construct this taxonomy (*e.g.* `PublicServant` – someone who works for the Government – might be modeled as a subclass of `Person`).

Functions are similar to relations, however, the last element of a function is unique given its preceding elements. For instance, a function `EvaluateEnterprise` applies a set of score rules to compute the final score an enterprise receives when participating in a specific procurement. For a given enterprise and procurement, there is exactly one score; therefore, the relationship is functional.

Formal axioms are used to model conditions that are always true in the domain. Axioms are commonly represented by first-order logic sentences. A common form for an

axiom is an if-then statement. For instance, if a person X is member of the committee responsible for procurement Y, then this person X cannot submit a proposal for this procurement Y. Axioms are often used for consistency checking and for inferring new knowledge [50].

2.1 UML and ER

Unified Modeling Language (UML) [115] and Entity/Relationship (ER) [24] diagrams are often used to organize information for community use. Some of the reasons to use these tools are [50]: (i) UML and ER are easy to understand; (ii) there are standard graphical representations for UML and ER diagrams; and (iii) many CASE tools are available to support development of UML and ER representations.

UML models can be enriched by adding Object Constraint Language (OCL) [3, 133] expressions. OCL is a declarative language for describing rules that apply to UML by providing expressions that do not have the ambiguities of natural language, and avoid the inherent difficulty of using complex mathematics.

In UML, classes are represented with rectangles divided into three parts: the name (top), the attributes (middle), and the operations (bottom). Since operations are not used in the context of the Semantic Web [50], we will not deal with them here. The attribute types, possible attribute values, and default values are included in their description. For instance, in Figure 2.1 `Person` is a class with attribute `name` of type `String`.

Instances of classes are represented as rectangles divided into two parts. The first part is the name of the instance, followed by “`:`” and the name of the class it represents. The second part contains the attributes of the instance and their respective values. For example, Figure 2.2 shows four instances, `winner1`, `participant2`, `participant3`, and `participant4`, of the class `Enterprise`.

Concept taxonomies are created through generalization relationships between classes. These are shown on a UML diagram by a solid line extending from the more specific to the more generic class, and ending with a large hollow triangle. In Figure 2.1, `Person` is a

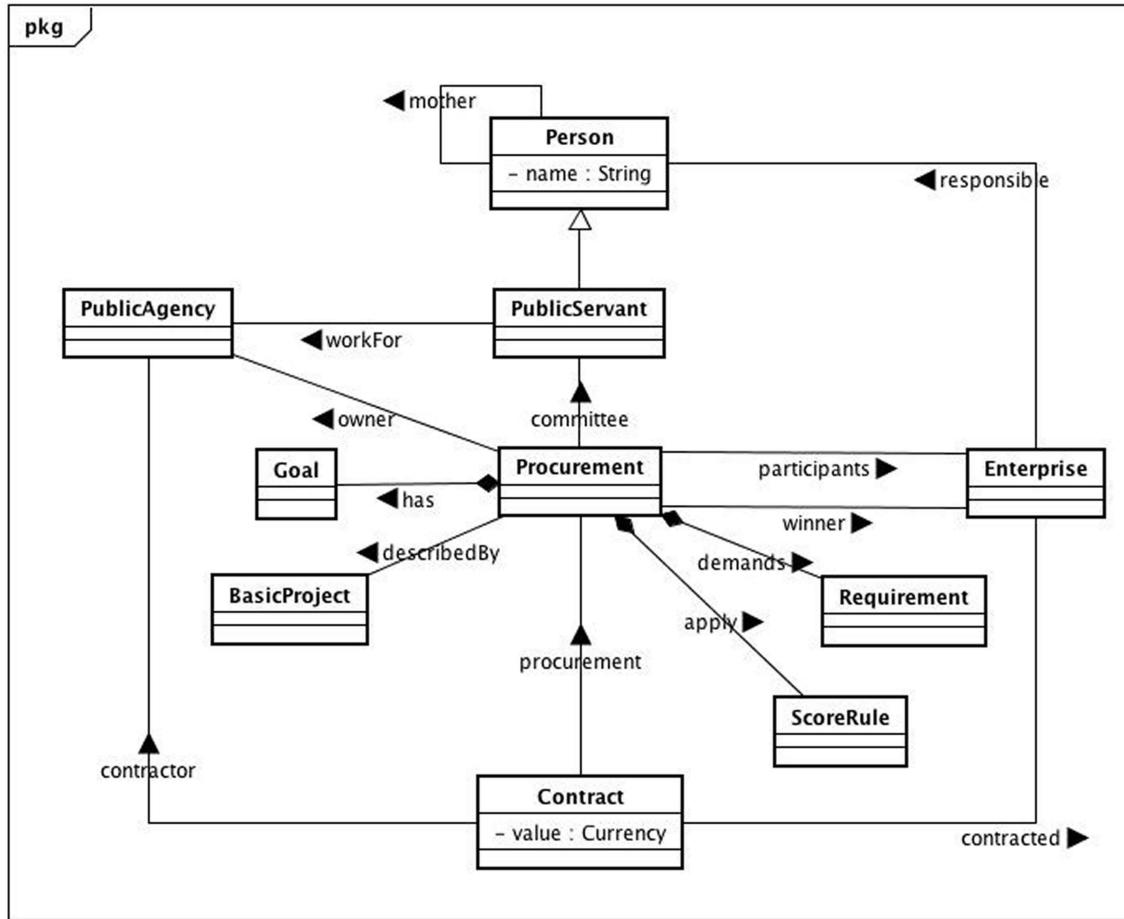


Figure 2.1: A class diagram for the procurement domain.

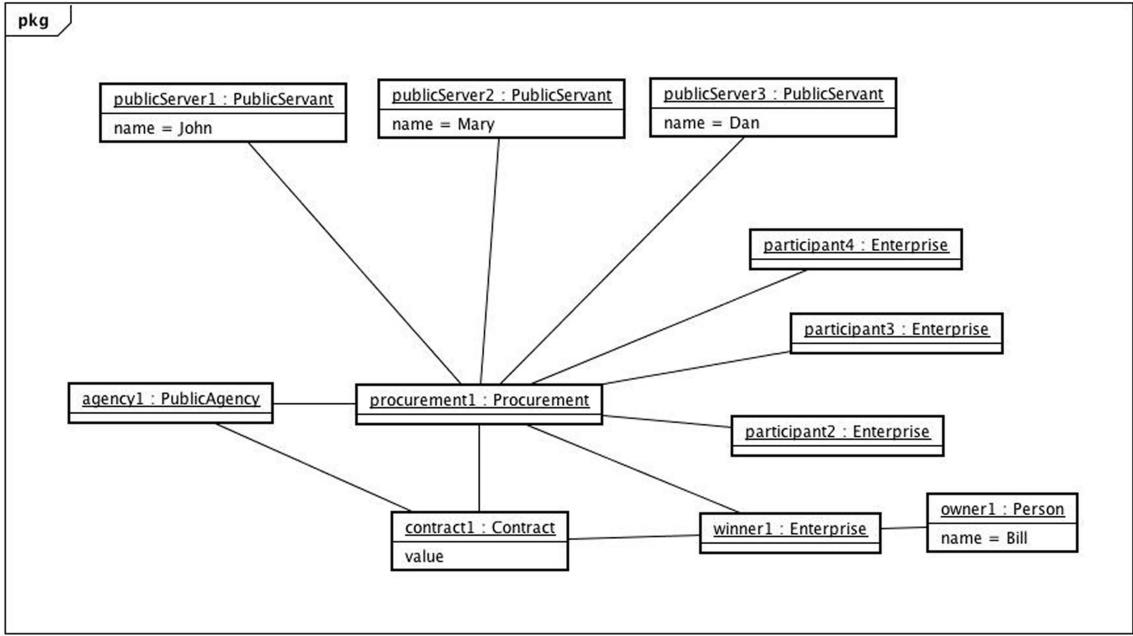


Figure 2.2: A class diagram with instances of classes from the procurement domain.

generalization of **PublicServant**, thus it inherits its attribute `name`.

Binary relations are expressed as associations (solid arrows) between classes. In Figure 2.1 a **PublicServant** works for a **PublicAgency**. However, higher arity relations cannot be represented directly in UML, though we can represent them by creating a class. This class is associated with other classes that represent the relation arguments, as shown in the ternary relation **Contract** also in Figure 2.1.

More complex modeling such as cardinalities of the attributes, disjoint and exhaustive knowledge, and formal axioms can be represented in UML only with the use of OCL. However, according to [50], there is no standard support for this language in common CASE tools. Because of this, and because UML models lack formal semantics, expressions in OCL cannot be evaluated by many CASE tools, and cannot be shared among developers.

In ER, with the common extension of generalization relationships between entities, it is possible to represent classes through the use of ER-entities. Furthermore, classes can be organized in taxonomies with the generalization relationship between ER-entities. For

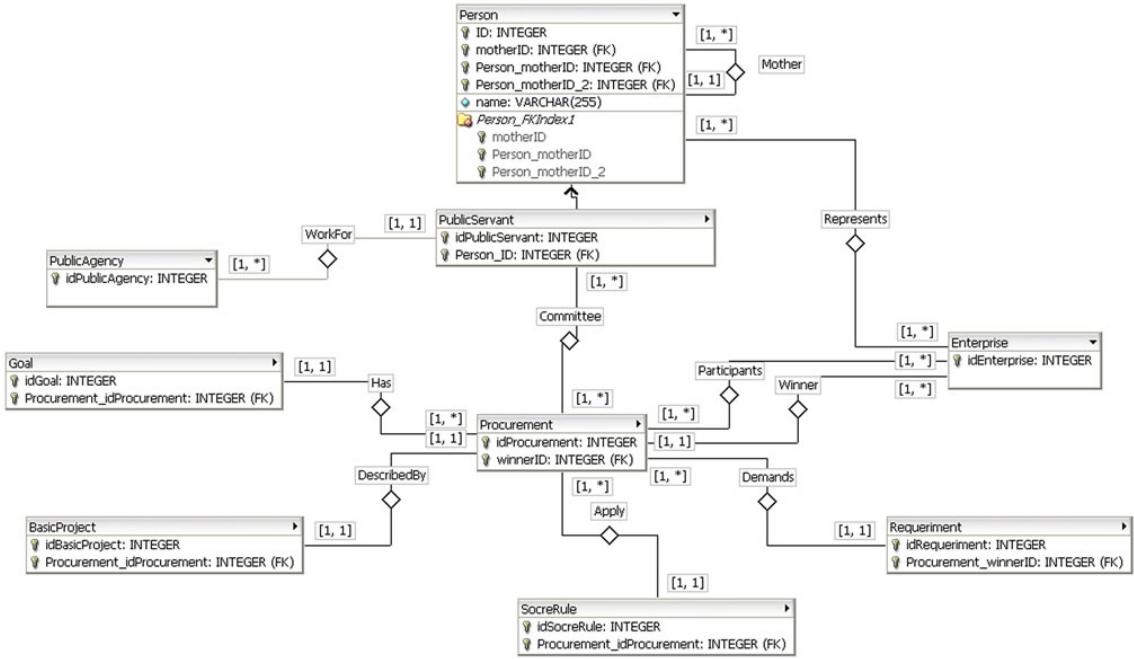


Figure 2.3: An extended entity-relationship diagram for the procurement domain.

example, Figure 2.3 shows the class **PublicServant**, which is a subclass of the class **Person**.

It is also possible to represent attributes and their types through ER-attributes. In Figure 2.3 the class **Person** has the attributes **name** and **ID** with types **VARCHAR(255)** and **INTEGER** respectively.

Ad hoc relations can be represented through ER-relations between ER-entities. These relations can have any arity and can have specified cardinalities. Figure 2.3 presents several relations. One of these is **Apply**, which relates one or more **ScoreRule** to just one **Procurement**. Although a ternary relation can be easily represented, none were included in Figure 2.3 for reasons of clarity.

Instances can be created through the use of the insert sentence from the Structured Query Language (SQL), which essentially becomes a filled row in its respective table.

According to [50], representing formal axioms in ER requires either extending the language, or using complementary notations, such as first-order logic or production rules.

2.2 Knowledge Representation and Reasoning

We have seen in Section 2.1 that UML and ER are able to represent classes, attributes, relations, and instances. However, they fall short when dealing with formal axioms. Knowledge representation systems, on the other hand, can naturally represent formal axioms. Furthermore, there are standard and readily available formal systems for reasoning with axioms.

According to [83] “Knowledge Representation is the field of study within AI concerned with using formal symbols to represent a collection of propositions believed by some putative agent.” In knowledge-based systems, the information believed by this putative agent is usually explicitly represented by a collection of symbolic structures, in other words, represented on the knowledge base (KB). Reasoning, on the other hand, is the process of manipulating these symbols in order to produce new information that is not explicitly represented in the KB.

To see the benefits of using Knowledge Representation and Reasoning (KR&R), we will augment the models presented in Section 2.1 by introducing axioms related to the domain. Based on Law N° 8,666/93, a member of a procurement committee must not be related to the enterprises that are participating in the procurement. For the sake of simplicity we will only deal with the relation that a public servant cannot be a brother/sister, son/daughter, or mother of someone who is responsible for that enterprise. Consider the following simple representation in Listing 2.1, expressed in standard first-order logic notation [42].

To be concise, we will not define the entire model. We assume the definitions are set up to reproduce the models from Section 2.1. Nevertheless, with the information presented above, we can identify an inconsistency. By combining lines 12, 13, 02, and 04, we can infer `Related(John,Bill)`. However, if we combine lines 15, 20, 19, and 05-06, we can infer `¬Related(John,Bill)`. Therefore, we have encountered an inconsistency in our KB. Fortunately, this representation allows us to debug, and to discover that this procurement is actually violating the law. We can then fix the inconsistency by modifying the rule in lines 05-06 to say that if a member of a procurement committee is related

to the enterprises that are participating in the procurement, then there is a violation of the law, *e.g.*, $\forall y \exists x, z, r \text{ Committee}(x, y) \wedge \text{Participant}(z, y) \wedge \text{Responsible}(r, z) \wedge \text{Related}(x, r) \rightarrow \text{ViolationOfLaw}(y)$.

Listing 2.1: Simple KR&R example in the procurement domain

1	$\forall x \text{ PublicServant}(x) \rightarrow \text{Person}(x)$
2	$\forall x, y, z \text{ Mother}(x, y) \wedge \text{Mother}(z, y) \rightarrow \text{Sibling}(x, z)$
3	$\forall x, y \text{ Mother}(x, y) \rightarrow \text{Related}(x, y)$
4	$\forall x, y \text{ Sibling}(x, y) \rightarrow \text{Related}(x, y)$
5	$\forall x, y, z, r \text{ Committee}(x, y) \wedge \text{Participant}(z, y) \wedge \text{Responsible}(r, z) \wedge \rightarrow \neg \text{Related}(x, r)$
6	$\text{PublicServant}(\text{John})$
7	$\text{PublicServant}(\text{Mary})$
8	$\text{PublicServer}(\text{Dan})$
9	$\text{Person}(\text{Bill})$
10	$\text{Person}(\text{Rebecca})$
11	$\text{Mother}(\text{John}, \text{Rebecca})$
12	$\text{Mother}(\text{Bill}, \text{Rebecca})$
13	$\text{Procurement}(\text{Procurement1})$
14	$\text{Committee}(\text{John}, \text{Procurement1})$
15	$\text{Committee}(\text{Mary}, \text{Procurement1})$
16	$\text{Committee}(\text{Dan}, \text{Procurement1})$
17	$\text{Enterprise}(\text{Winner1})$
18	$\text{Responsible}(\text{Bill}, \text{Winner1})$
19	$\text{Participant}(\text{Winner1}, \text{Procurement1})$

One could argue that this restriction can be easily implemented by adding an operation `isCommitteeRelatedToParticipants()` to the class `Procurement` from our UML model in Section 2.1, for instance. This operation would return `true` if there is a relation, as defined above, between one of the members of the committee and one of the responsible persons of the enterprises that participates in the procurement. However, UML lacks a formal way to define such an operation in detail, leaving its implementation open to the implementer.

This has at least two main disadvantages. First, every system that uses this model has to implement its own interpretation of the operation. In addition to creating duplication of effort, this could easily lead to differing interpretations and inconsistent implementations. Second, if for some reason the rule changes (*e.g.*, we realize we need to include father in our relation), then every interpretation of the model, *i.e.* every system that uses this model, would have to change its implementation, instead of just changing the model as we would do in a knowledge-based system. This simple example illustrates several advantages of using KR&R in modeling.

One of the main advantages of knowledge-based systems is that they take action based on what our putative agent believes, and not only on what is explicitly represented in the KB. We could easily see that in the example given above where the information inferred by the system was crucial in finding an infringement of the law.

In fact, the inference process described above is entailment, which is exactly what makes logic an important element of KR&R, since logic is the study of entailment relations. According to [83, 11], entailment is defined as:

Definition 2.1. We say that the propositions represented by a set of sentences S entail the proposition represented by a sentence p when the truth of p is implicit in the truth of the sentences in S . In other words, entailment means that if the world is such that every element of S comes out `true`, then p does as well.

According to [83, 11], a knowledge-based system can be seen as a system that performs some problem-solving activity. An example of such activity is verifying whether there is a member of the committee who is related to one of the responsible persons of an enterprise that participates in a specific procurement. It is able to do so by looking at what it already knows: Is a member mother of a responsible person of a participant enterprise? Does a member have the same mother as a responsible person of a participant enterprise? In order to answer such questions the system has to look at the explicit information available in its KB and make inferences based on it. Therefore, it is reasonable to separate the representation and management of the KB from the rest of the system. In other words,

it is not important for a problem-solving system how the knowledge is stored or how the reasoning is done to produce new knowledge. This kind of system is only concerned with answering its domain-specific questions, which in this case would be to answer questions related to fraud in public procurements.

According to [83,11], “It is the role of a knowledge representation system to manage the KB within a larger knowledge-based system. Its job is to make various sorts of information about the world available to the rest of the system based on the information it has obtained, perhaps from other parts of the system, and by using whatever reasoning it can perform. Therefore, the job of the KR system is smaller than that of a full knowledge-based problem solver, but larger than that of a database management system, which would merely retrieve the contents of the KB.”

2.3 Ontology and the Semantic Web

According to [10] the Semantic Web is a web of data that can be processed directly or indirectly by machines. This technology will drive us to a new phase where the arduous and manual task of identifying, accessing and utilizing information is assigned to computers, allowing human beings to change their focus from data driven to knowledge driven activities.

The W3C [60] states that ontologies are envisioned as the technology providing the cement for building the SW. The term ontology was taken from Philosophy, where it means a systematic explanation of being. In the field of knowledge representation, an ontology contains a common set of terms for describing and representing a domain in a way that allows automated tools to use stored data in a context-aware fashion, intelligent software agents to afford better knowledge management, and many other possibilities brought by a standardized, more intensive use of meta-data. [127] defines ontology as:

Definition 2.2. An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon in the world by having identified the relevant concepts of that phenomenon. Explicit means that the type

of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group.

Since ontologies are used in different communities and with different objectives, [130] provided a new definition to make it more popular in all these different disciplines:

Definition 2.3. An ontology may take a variety of forms, but it will necessarily include a vocabulary of terms and some specification of their meaning. This includes definitions and an indication of how concepts are inter-related which collectively impose a structure on the domain and constrain the possible interpretation of terms.

According to [50] there are basically two distinct classes of ontologies: lightweight and heavyweight. The first describes the main concepts, their attributes and relationships while the second adds axioms that make the possible valid interpretations closer to the true intended meaning.

An ontology is used to define the vocabulary that will be shared between different agents. The agreement of using and accepting the definitions in this vocabulary is called ontological commitment [54]. The main idea behind this shared vocabulary is to allow different agents, that have different information about the domain of discourse, to make assertions about their knowledge and ask queries to other agents about information they might be interested in.

Next, I will show the advantages of using ontology and the semantic web, then I will present the origin and an overview of Web Ontology Language (OWL) which is the basis for the probabilistic web ontology language, PR-OWL, I am extending in this dissertation.

2.3.1 The Advantages of Ontology and the Semantic Web

How do Ontology and the SW differ from what we have seen that UML, ER, and knowledge-based KR&R systems can model? Well, as seen before, the SW is designed for RIS environments, which are characterized by the AAA slogan, the open world assumption, and

nonunique naming. But this style of information gathering can create a chaotic landscape rife with confusion, disagreement and conflict. UML, ER, and knowledge-based KR&R systems were developed under a more constrained paradigm for information sharing, and lack some important features needed to contain the chaos to which RIS environments are prone. *E.g.*, there is no formal way to allow anyone to say anything about any topic (AAA slogan) on the web in these languages. A number of SW modeling languages have been developed expressly for the RIS environments. These languages differ in their capabilities and their level of expressivity, but all incorporate features necessary to foster cooperative and collaborative information sharing in RIS environments.

It is easy to see that our domain of fraud detection/prevention is a RIS environment. The data CGU has available does not come only from its audits and inspections. In fact, much complementary information can be retrieved from other Federal Agencies, including Federal Revenue Agency, Federal Police, and others. Imagine we have information about the enterprise that won the procurement, and we want to know information about its owners, such as their personal data and annual income. This type of information is not available at CGUs Data Base (DB), but must be retrieved from the Federal Revenue Agency's DB. Once the information about the owners is available, it might be useful to check their criminal history. For that (see Figure 2.4), information from the Federal Police must be used. In this example, we have different sources saying different things about the same person: thus, the AAA slogan applies. Moreover, there might be other Agencies with crucial information related to our person of interest; in other words, we are operating in an open world. Finally, to make this sharing and integration process possible, we have to make sure we are talking about the same person, who may (especially in case of fraud) be known by different names in different contexts.

2.3.2 The Beginning of OWL

The Web Ontology Language (OWL) was developed by the World Wide Web Consortium (W3C) Web Ontology Working Group and is an effort in W3C's Semantic Web activity.



Figure 2.4: Retrieving information through the SW for the procurement domain.

As such, it had to be compatible with the vision of a stack of languages including XML and RDF. Besides that, by the time the W3C started defining OWL specification, there were already some other languages designed to be used on the Web, including OIL [43] and DAML+OIL [65].

In fact, the main influences in OWL's design were [66] the DAML+OIL language, the Description Logics (DL), the frames paradigm, and the RDF language, a requirement for upwards compatibility which lead to the RDF/XML exchange syntax.

The frames paradigm, proposed by Minsky [92], is a knowledge representation scheme that organizes knowledge into chunks called frames. These frames should described the main ideas contained in some typical situation, for example participating in a procurement or analyzing the proposals, by putting all relevant information for these situations together. Collections of interconnected frames are then organized in frame systems.

According to [44], features that are common to frame-based systems are:

- Frames are organized in (tangled) hierarchies;
- Frames are composed out of slots (attributes) for which fillers (scalar values, references to other frames or procedures) have to be specified or computed; and
- Properties (fillers, restriction on fillers, etc.) are inherited from **superframes** to **subframes** in the hierarchy according to some inheritance strategy.

These organizational principles turned out to be very useful, and, indeed, the now popular object-oriented languages have adopted these organizational principles. For the same reason, the surface structure of the OWL language (as seen in the abstract syntax) was influenced by the frames paradigm.

According to [8], Description Logics are a family of languages which allow the representation of domain-specific knowledge in a structured and formal way. Its name comes from the fact that important notions of the domain are described by concept descriptions and the fact that it is equipped with a formal, logic-based semantics.

The Semantic Web depends on the availability of a well-defined semantics and powerful reasoning tools, which are both provided by Description Logics, that is why OWL's formal specification was particularly influenced by Description Logics [66].

DAML+OIL [65] is the result of a merger between DAML-ONT, a language developed as part of the US DARPA Agent Markup Language (DAML) program) and the Ontology Inference Layer (OIL) [43], developed by a group of (mostly) European researchers.

DAML+OIL is based on common ontological primitives from Frame languages, which makes it human understandable. Its syntax is based on RDF Schema, which provides web compatibility. Finally, its semantics can be defined by a translation into the expressive DL SHIQ [67], a DL language that is decidable, even though its worst-case complexity is exponential. Nevertheless, in practice it does behave quite well [8].

2.3.3 The Web Ontology Language (OWL)

The main concepts available in OWL are [89]:

Class A class defines a group of individuals that belong together because they share some properties;

rdfs:subClassOf Class hierarchies may be created by making one or more statements that a class is a subclass of another class;

rdf:Property Properties can be used to state relationships between individuals or from individuals to data values;

rdfs:subPropertyOf Property hierarchies may be created by making one or more statements that a property is a subproperty of one or more other properties;

rdfs:domain A domain of a property limits the individuals to which the property can be applied;

rdfs:range The range of a property limits the individuals that the property may have as its value; and

Individual Individuals are instances of classes, and properties may be used to relate one individual to another.

Most of these basic terms, except **rdfs:subPropertyOf**, can also be represented in languages like UML and ER diagrams. The expressive power of OWL comes from features like defining classes as complex construction using other classes and restriction on properties.

An example of a simple class definition using intersection of other classes is the class **CorruptAgent**, which can be defined as the intersection of **PublicServant** and **CorruptPerson**. In OWL this would be defined as shown in Listing 2.2.

A more complex example, which uses restriction on properties, is the class **Mother**, which can be defined as a **Person** of female sex that has at least one child. In OWL this would be defined as shown in Listing 2.3.

Listing 2.2: Definition of the class `CorruptAgent` in OWL

```

1 <owl:Class rdf:about="CorruptAgent">
2   <owl:equivalentClass>
3     <owl:Class>
4       <owl:intersectionOf rdf:parseType="Collection">
5         <rdf:Description rdf:about="CorruptPerson"/>
6         <rdf:Description rdf:about="PublicServant"/>
7       </owl:intersectionOf>
8     </owl:Class>
9   </owl:equivalentClass>
10 </owl:Class>

```

These are just a few of the more expressive representations that OWL allows, which cannot be represented in languages like UML and ER diagrams. OWL also provides vocabulary for (in)equality, property characteristics and restrictions, cardinality, Boolean operators, etc. For more details on the OWL's syntax and semantics see [123].

Listing 2.3: Definition of the class `Mother` in OWL

```

1 <owl:Class rdf:about="Mother">
2   <owl:equivalentClass>
3     <owl:Class>
4       <owl:intersectionOf rdf:parseType="Collection">
5         <rdf:Description rdf:about="Person"/>
6         <owl:Restriction>
7           <owl:onProperty rdf:resource="hasChild"/>
8           <owl:someValuesFrom rdf:resource="Person"/>
9         </owl:Restriction>
10        <owl:Restriction>
11          <owl:onProperty rdf:resource="hasSex"/>
12          <owl:hasValue rdf:resource="female"/>
13        </owl:Restriction>
14      </owl:intersectionOf>
15    </owl:Class>
16  </owl:equivalentClass>
17 </owl:Class>

```

In 2009 a new version of OWL was proposed for recommendation [53]. OWL 2 is quite similar to its previous version. The central role of RDF/XML, the role of other syntaxes, and the relationships between the Direct and RDF-Based semantics (*i.e.*, the correspondence theorem) have not changed. More importantly, backwards compatibility with OWL 1 is, for

all intents and purposes, complete: all OWL 1 Ontologies remain valid OWL 2 Ontologies, with identical inferences in all practical cases.

OWL 2 adds new functionality with respect to OWL 1. Some of the new features are syntactic sugar (*e.g.*, disjoint union of classes) while others offer new expressivity, including:

- Keys;
- Property chains;
- Richer datatypes, data ranges;
- Qualified cardinality restrictions;
- Asymmetric, reflexive, and disjoint properties; and
- Enhanced annotation capabilities.

For more information about OWL 2 see [53].

Chapter 3: Representing Uncertainty

Consider the example from Section 2.2, in which it is defined that a member of the procurement must not be related to a person responsible for an enterprise that is participating in the same procurement. Current SW deterministic reasoning algorithms will either consider this relation to be true, false, or unknown, with no way of expressing gradations of plausibility.

This is acceptable in situations where complete information is available. However, in open world environments such as the Web, partial (not complete) or approximate (not exact) information is more the rule than the exception. For example, we may not have the information from lines 11 and 12 from Section 2.2 stating that **John** and **Bill** have the same mother, **Rebecca**. However, we do have information about **John** and **Bill** stating that they have a common last name and live at the same address. Although we are uncertain about whether or how they are related, there is evidence suggesting they are. It is important to consider that information when reasoning about possible violations of procurement regulations.

Although the above and similar examples imply the need for principled representation and reasoning with uncertainty within the SW, current SW applications (including current automated reasoning methods) are primarily based on classical logic. This includes OWL, the W3C standard web ontology language, which has its logical basis in classical description logic, and therefore lacks built-in support for uncertainty. This is a major shortcoming for a technology intended to operate in RIS environments. The W3C responded to this limitation by initiating the Uncertainty Reasoning for the World Wide Web Incubator group (URW3-XG), created in 2007 and concluded a year later. The URW3-XG concluded that standardized representations were needed to express uncertainty in Web-based information [82].

Uncertainty is especially important to applications such as corruption prevention, in which perpetrators seek to conceal illicit intentions and activities. To address the SW lack of support for uncertainty, Costa [27] proposed a Bayesian framework for probabilistic ontologies. Probabilistic ontologies have the expressiveness required for SW applications, and yet provide a principled logical basis for representing and reasoning under uncertainty. The probabilistic ontology language PR-OWL [27, 30] is based on Multi-Entity Bayesian Networks (MEBN) [78, 76] a probabilistic logic that combines the expressive power of First-Order Logic (FOL) with Bayesian networks’ ability to perform plausible reasoning.

3.1 Multi-Entity Bayesian Network (MEBN)

Multi-Entity Bayesian Networks (MEBN) extend Bayesian Networks (BN) to achieve first-order expressive power. MEBN represents knowledge as a collection of MEBN Fragments (MFrags), which are organized into MEBN Theories (MTheories).

An Mfrag contains random variables (RVs) and a fragment graph representing dependencies among these RVs. It represents a repeatable pattern of knowledge that can be instantiated as many times as needed to form a BN addressing a specific situation, and thus can be seen as a template for building and combining fragments of a Bayesian network. It is instantiated by binding its arguments to domain entity identifiers to create instances of its RVs. There are three kinds of nodes: context, resident and input. Context nodes represent conditions that must be satisfied for the distributions represented in the Mfrag to apply. Input nodes may influence the distributions of other nodes in an Mfrag, but their distributions are defined in their home MFrags. Distributions for resident nodes are defined within the Mfrag by specifying local distributions conditioned on the values of the instances of their parents in the fragment graph.

A set of MFrags represents a joint distribution over instances of its random variables. MEBN provides a compact way to represent repeated structures in a BN. An important advantage of MEBN is that there is no fixed limit on the number of RV instances, and the

random variable instances are dynamically instantiated as needed.

An MTheory is a set of MF frags that collectively satisfy conditions of consistency ensuring the existence of a unique joint probability distribution over its random variable instances.

To apply an MTheory to reason about particular scenarios, one needs to provide the system with specific information about the individual entity instances involved in the scenario. Upon receipt of this information, Bayesian inference can be used both to answer specific questions of interest (*e.g.*, how likely is it that a particular procurement is being directed to a specific enterprise?) and to refine the MTheory (*e.g.*, each new situation includes additional data about the likelihood of fraud for that set of circumstances). Bayesian inference is used to perform both problem specific inference and learning from data in a sound, logically coherent manner.

3.2 Probabilistic Web Ontology Language (PR-OWL)

The common approach to representing uncertainty in knowledge representation languages where uncertainty has been introduced as an afterthought, is to use simple XML tags to represent a number between 0 and 1, which is the probability value. However, this is just one aspect of probabilities and according to various authors, not the most relevant one. In fact, researchers have stated in many instances the importance of structural information in probabilistic models (see [119]). For instance, [121] stated that probability is more about structure than it is about numbers.

PR-OWL is a language for representing probabilistic ontologies. Probabilistic ontologies go beyond simply annotating ontologies with probabilities to provide a means of expressing subtle features required to express a first-order Bayesian theory. Because PR-OWL is based on MEBN logic, it not only provides a consistent representation of uncertain knowledge that can be reused by different probabilistic systems, but also allows applications to perform plausible reasoning with that knowledge, in an efficient way. Work on PR-OWL is based on the following definition of a probabilistic ontology [27]:

Definition 3.1. A probabilistic ontology is an explicit, formal knowledge representation that expresses knowledge about a domain of application. This includes:

1. Types of entities existing in the domain;
2. Properties of those entities;
3. Relationships among entities;
4. Processes and events that happen with those entities;
5. Statistical regularities that characterize the domain;
6. Inconclusive, ambiguous, incomplete, unreliable, and dissonant knowledge;
7. Uncertainty about all the above forms of knowledge;

where the term entity refers to any concept (real or fictitious, concrete or abstract) that can be described and reasoned about within the domain of application.

Probabilistic ontologies are used for the purpose of comprehensively describing knowledge about a domain and the uncertainty associated with that knowledge in a principled, structured, and sharable way. PR-OWL was developed as an extension enabling OWL ontologies to represent complex Bayesian probabilistic models in a way that is flexible enough to be used by diverse Bayesian probabilistic tools based on different probabilistic technologies (*e.g.* PRMs, BNs, etc). More specifically, PR-OWL is an upper ontology (*i.e.* an ontology that represents fundamental concepts that cross disciplines and applications) for probabilistic systems. PR-OWL is expressive enough to represent even the most complex probabilistic models. It consists of a set of classes, subclasses and properties that collectively form a framework for building probabilistic ontologies.

OWL 1 has three different versions with increasing expressive power designed for specific communities of developers and users. The least expressive version is OWL Lite, which has a limited set of simple restrictions. The next step in expressiveness in the OWL family is OWL DL, which is based on Descriptive Logic and aims to maximize expressiveness

while maintaining completeness (all logical consequences are provable) and decidability (all proofs terminate in finite time). OWL-DL has all OWL constructions, but there are certain restrictions on use. The most expressive version, OWL Full, was built for users who want the strongest representational power possible in OWL format. As a consequence, there are no guarantees of computability. Following the same reasoning, a PR-OWL Lite version could be created as suggested in [27] by including some restrictions.

OWL 2, on the other hand, introduces the concept of profiles. According to [53], profiles are sub-languages of OWL 2. As such, they are more restrictive than OWL DL. Three profiles are described in OWL 2: OWL EL, OWL QL, and OWL RL. OWL EL is intended for applications that use large ontologies with focus on performance. OWL QL is more suitable to lightweight ontologies that need to access data via relational queries, like SQL. Finally, OWL RL is recommended for lightweight ontologies that need to access and manipulate data in the form of RDF triples.

PR-OWL was proposed as an extension to the OWL language based on MEBN, which can express a probability distribution on interpretations of any first-order theory. As a consequence, there are no guarantees that reasoning with PR-OWL ontology will be efficient or even decidable [27]. For problems in which computational efficiency is a concern, well-known classes of computationally efficient Bayesian theories can be represented in PR-OWL. PR-OWL was built to be interoperable with non-probabilistic ontologies. Since PR-OWL adds new definitions to OWL while retaining backward compatibility with its base language, OWL-built legacy ontologies will be able to interoperate with newly developed probabilistic ontologies. However, the ontology's probabilistic definitions have to form a valid complete or partial MTheory. Figure 3.1 (from [27] page 149) shows the main concepts involved in defining an MTheory in PR-OWL.

In the diagram, ellipses represent general classes while arrows represent the main relationships between these classes. A probabilistic ontology (PO) has to have at least one individual of class MTheory, which is basically a label linking a group of MFrags that collectively form a valid MTheory. In actual PR-OWL syntax, that link is expressed via the

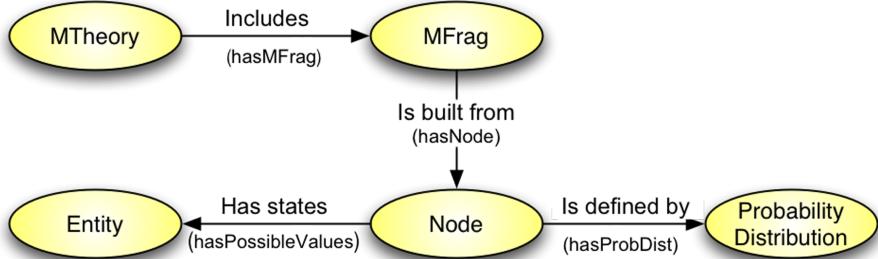


Figure 3.1: PR-OWL simple model (reproduced with permission from Costa [27]).

object property `hasMfrag` (which is the inverse of object property `isMfragIn`). Individuals of class `MFragment` are comprised of nodes (not shown in the picture). Each individual of class `Node` is a random variable (RV) and thus has a mutually exclusive and collectively exhaustive set of possible states. In PR-OWL, the object property `hasPossibleValues` links each node with its possible states, which are individuals of class `Entity`. Finally, random variables (represented by the class `Node` in PR-OWL) have unconditional or conditional probability distributions, which are represented by class `ProbabilityDistribution` and linked to their respective nodes via the object property `hasProbDist`.

Figure 3.2 (from [27] page 150) depicts the main elements of the PR-OWL language, its subclasses, and the secondary elements necessary for representing an `MTheory`. The relations necessary to express the complex structure of MEBN probabilistic models using the OWL syntax are also depicted.

The first step towards building a probabilistic ontology as defined above is to import the PR-OWL ontology into an ontology editor (*e.g.*, OntoEdit, Protégé, Swoop, etc.) and start constructing the domain-specific concepts using the PR-OWL definitions to represent uncertainty about their attributes and relationships. Using this procedure, a knowledge engineer is not only able to build a coherent `MTheory` and other probabilistic ontology elements, but also make it compatible with other ontologies that use PR-OWL concepts. However, building MFrags this way is a manual, error prone, and tedious process that requires deep knowledge of the logic and of the data structures of PR-OWL in order to

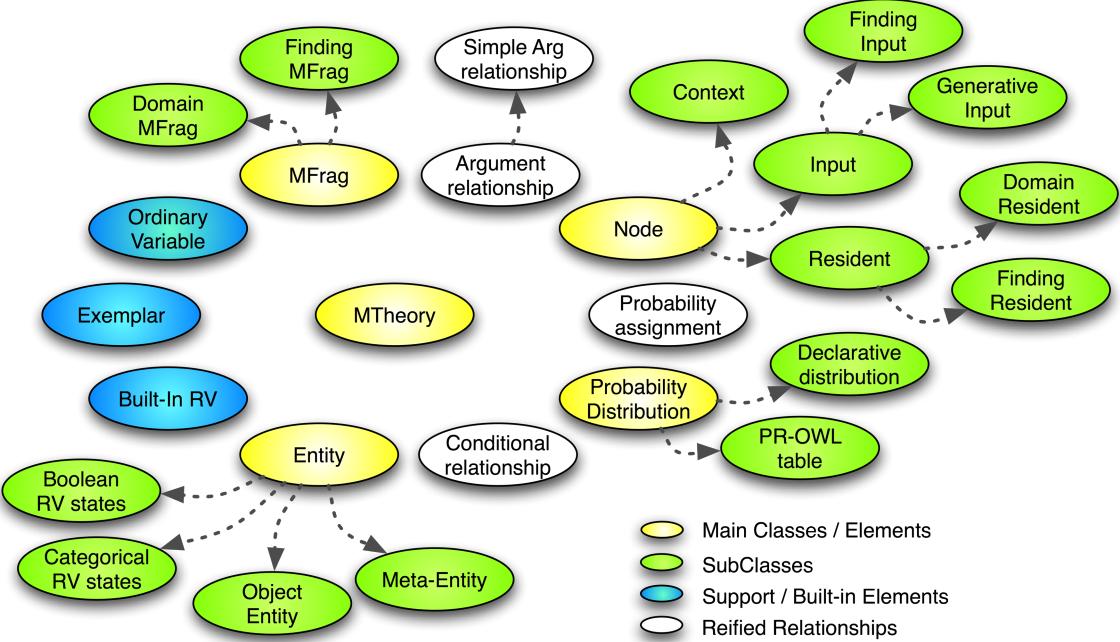


Figure 3.2: PR-OWL detailed model (reproduced with permission from Costa [27]).

avoid errors or inconsistencies. UnBBayes [2] changes all that by providing a GUI-based editing process for building probabilistic ontologies based on the PR-OWL upper ontology for probabilistic theories [23]. Another important feature is the ability to save and open models created by the UnBBayes GUI in PR-OWL format, with backwards compatibility to OWL through the use of the Protégé API. Protégé [1] is an ontology editor and a flexible and configurable framework for building knowledge-based tools and applications. Protégé was developed by the Stanford Center for Biomedical Medical Informatics Research.

The major advantages of using PR-OWL are its flexibility and representational power, both inherited from the fact that the language is based on MEBN, a full integration of First-Order Logic (FOL) and probability theory that merges the expressiveness of the former with the inferential power of the latter. UnBBayes leverages this power with a built-in MEBN reasoner [26]. The prospective reader can find additional details on PR-OWL at <http://www.pr-owl.org>.

3.3 Related Work

The past few years have witnessed an increasingly mature body of research on the Semantic Web, with new standards being developed and more complex use cases being proposed and explored. As complexity increases in SW applications, so does the need for principled means to cope with uncertainty inherent to real world SW applications. Not surprisingly, several approaches addressing uncertainty representation and reasoning in the Semantic Web have emerged. This Section will give a brief overview on those different approaches and how they relate to the work being proposed in PR-OWL 2.

3.3.1 First-Order Probabilistic Languages (FOPL)

Just as propositional logic is not enough to represent most of the real-world applications in the SW, propositional probabilistic models, such as Bayesian networks (BN) and Markov networks (MN) are insufficient to leverage uncertainty reasoning in the SW. First, they do not represent repeated structure, such as uncertainty about an attribute that applies to all instances of a given class. Second, they define probability distributions only for a fixed and predefined set of random variables.

Therefore, the need for more expressive probabilistic models becomes evident. This role can be filled by first-order probabilistic languages (FOPLs), which are languages that can model large families of random variables compactly by abstracting over objects, the same way first-order logic leverages propositional logic [91].

Over the last few years, a plethora of FOPL have been proposed. To better understand their similarities and differences, Milch and Russell [91] proposed a taxonomy (see Figure 3.3 from [91] page 3), which will be described next.

The first characteristic to distinguish the different FOPLs is their outcome spaces, in other words, the sets of outcomes to which they assign probabilities. The most common outcome space is a set of relational structures, which are similar to graphs (as seen in BNs and MNs) where the nodes represent objects in the domain and the edges represent

probabilistic relations.

According to Milch and Russell [91] one reason for the variety of FOPLs is the different interpretations associated to relational structures. In logic, a relational structure is represented by the domain of discourse and an interpretation of the language over the domain. Another interpretation of relational structures can be the thought of as instances of a relational database schema. In statistics, possible outcomes are seen as instantiations of a set of random variables.

Examples of languages that follow the logic view, where a distribution is defined over the logical model structure, include, amongst others, Multi-Entity Bayesian Networks (MEBNs) [78], Relational Bayesian Networks (RBNs) [69], Bayesian Logic (BLOG) [90], and Markov Logic Networks (MLNs) [40]. Probabilistic Relation Models (PRMs) [107] is one of the distinct languages that follow the database view. Finally, the statistical view on relational structure is the basis for the plates model underlying Bayesian inference Using Gibbs Sampling (BUGS) [125].

Even though they have different views on their outcome space, they all fall into the category of relational structures, as Milch and Russell [91] have defined it. Two distinct languages that have different outcome spaces are Stochastic Logic Programs (SLPs) [99] and IBAL [106].

On the one hand, SLPs define a distribution over proofs from a given logic program. If a particular goal predicate R is specified, then an SLP also defines a distribution over tuples of logical terms: the probability of a tuple (t_1, \dots, t_k) is the sum of the probabilities of proofs of $R(t_1, \dots, t_k)$.

On the other hand, IBAL is a general-purpose programming language with stochastic choices, where distributions are defined over environments that map symbols to values. Although these values can be the same as states of random variables in BNs, for instance, they can also be other environments, or even functions.

The second characteristic used to define the taxonomy for FOPLs is the specificity of the language. Amongst the languages that have relational structures as their outcome

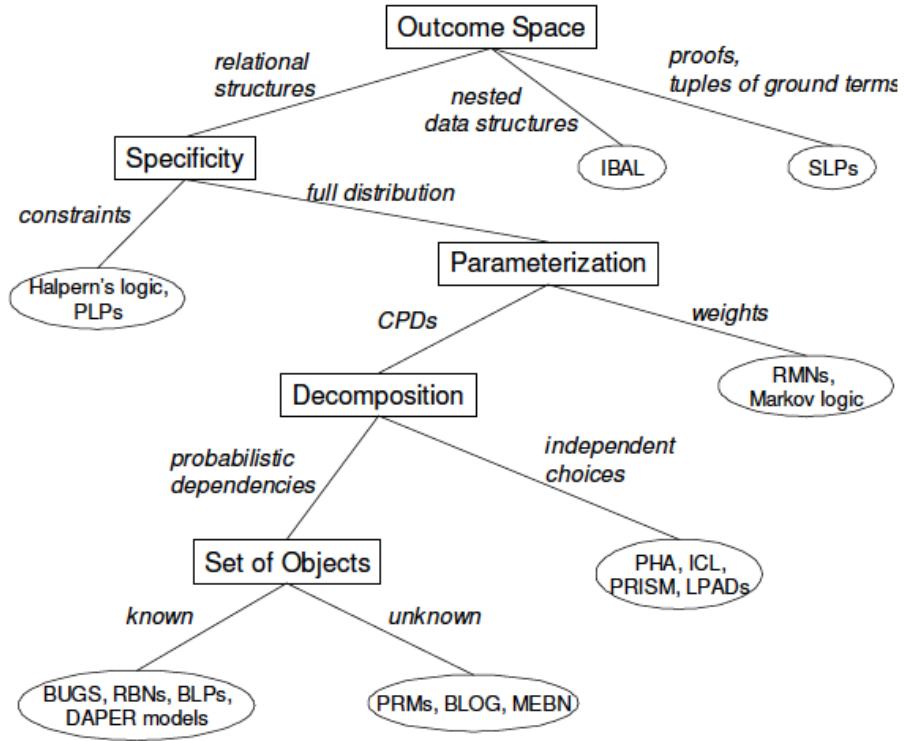


Figure 3.3: A taxonomy of first-order probabilistic languages (reproduced with permission from Milch and Russell [91]).

space, most of them fully define a distribution and some of them only impose constraints on a distribution. Halpern’s logic of probability on possible worlds [58] is an example of languages that generally do not fully define a distribution, instead, they usually define particular marginal probabilities.

The third characteristic taken into consideration on the taxonomy from Figure 3.3 is the parameterization, which is basically broken down into two categories, those based on Bayesian Networks (BNs), which use conditional probability distributions (CPDs) and those based on Markov Networks (MNs), which use weights to define the relative probabilities of instantiations.

As expected, there is a trade-off between FOPLs that use CPDs and the ones that use weights. The trade-off comes from the trade-off already well-known in the literature comparing BNs and MNs (see [105] Chapter 3 for details). Examples of languages that uses

CPDs are MEBN [78], PRM [107], and BLOG [90], while MLN [40] is an example of FOPL that uses weights. The following is a summary of this trade-off from [91].

On the one hand languages using CPDs according to Milch and Russell [91] have several advantages:

... the parameters have clear interpretations as prior and conditional probabilities, and can be estimated from fully observed data using elementary formulas. Even more importantly, the parameters are modular: they reflect causal processes that apply regardless of the relational skeleton¹.

However, one major drawback is that acyclic relations are not allowed, which can be hard to impose on every possible relational structure. Besides, some probabilistic relations are just hard to represent using directed models (*e.g.* BNs) and are more naturally represented in undirected models (*e.g.* MN). Milch and Russell [91] present some specific examples on the pros and cons of both approaches.

On the other hand, in languages using weights, there is no acyclic restriction. However, parameters for these models cannot be found using simple formulas, and according to Milch and Russell [91]:

... we need to ensure that the relational skeletons in the training set reflect the diversity of relational skeletons that may be encountered in the test data.

For more information on this characteristic of conditional probabilities versus weights, see Section 2.3 in [91].

The fourth characteristic on the taxonomy of FOPL is the decomposition. Some languages stand out since they restrict their CPDs to be only deterministic and allow probabilistic assignments only to variables with no parents. Examples of languages that impose this restriction are PRISM [117,118], probabilistic Horn abduction [108], independent choice logic (ICL) [109], and Logic Programs with Annotated disjunctions (LPADs) [131].

¹A relational skeleton is a partial specification of a database instance with values specified for all primary and foreign keys. However, the attribute values are left unspecified [48].

Finally, the last characteristic is whether the language supports unknown objects, or if it requires the set of objects to be specified in the relational skeleton. Only three languages have unknown objects as a fundamental concept present on their semantics, PRM [107], BLOG [90], and MEBN [78]. All the others assume that objects are in one-to-one correspondence with a given set of constant symbols, or grounding of the language. This feature is especially important in domains where reports about objects are received, but there is no way of knowing in advance if this object is already present in our knowledge base or if it is a new one.

3.3.2 Probabilistic Languages for the SW

In the past few years, as the Semantic Web community has developed standards and more complex use cases, appreciation has grown of the need for principled approaches for representing and reasoning under uncertainty. As a consequence, several approaches to uncertainty reasoning for the Semantic Web (SW) have emerged [27, 37, 61, 71, 85, 103, 126, 128]. This Section will focus on approaches that use probabilistic methods for representing uncertainty on the SW. The main reason for choosing these approaches is that, as stated by Predoiu and Stuckenschmidt [113], probabilistic methods are a natural choice for plausible reasoning on the SW, since great benefit could come for the SW with a tight integration with the machine learning and information retrieval techniques, which in most cases are based on probabilistic methods.

These languages can be separated into four different groups [113], extensions of RDF, extensions of OWL, extensions of Description Logics (DL), and extensions of Logic Programming formalisms.

The first group is basically represented by Fukushige [46], who proposes a vocabulary for representing elements of a Bayesian Network (BN) in RDF and link them to regular RDF triples, and by Udrea *et. al.* [129], who propose pRDF, a formal probabilistic extension of RDF, which can be considered a probabilistic logic on its own [113].

Examples of languages in the second group are PR-OWL [27], described in Section 3.2,

as well as OntoBayes [136], a language capable of representing random variables and their probabilistic dependencies through conditional probability distributions. Also in the same group, there is the work of Holi and Hyvönen [63], who propose a framework for representing uncertainty on taxonomies using BNs, and BayesOWL [37], a more expressive version of [63] for representing uncertainty about class memberships within OWL ontologies.

In the third group, two main approaches have been proposed for extending Description Logics (DLs) with probabilistic information. On the one hand, there is the work of Lukasiewicz [85]², who proposes probabilistic extensions of SHIF(**D**) and SHOIN(**D**), that allow expressing both terminological probabilistic knowledge about concepts and roles as well as assertional probabilistic knowledge about instances of concepts and roles based on probability intervals. On the other hand, there is P-CLASSIC, proposed by Koller *et. al.* [71], an approach that uses a complete specification of the probability distribution using BNs where the nodes are associated to concept expressions in the CLASSIC DL.

Finally, in the fourth group, there is a group of languages that are of special interest to the SW community [113], those that involve ideas about how to connect rule bases with ontologies represented in OWL or related languages. These languages are basically divided into two groups [113]: those that integrate OWL with Logic Programming by allowing to specify both a logic program and a description logic knowledge base and allowing them to interact; and those based on Description Logic Programs (DLP) [52] and on a translation from OWL to Logic Programming formalisms that have been extended to deal with uncertainty. Examples of the first include Lukasiewicz [84] and Cali *et. al.* [15], while the second include the work of Predoiu [111], Predoiu and Stuckenschmidt [112], and Nottelman and Fuhr [100].

²This paper is an revised and extended version of [49].

Chapter 4: A Formal Definition for Probabilistic Ontology - PR-OWL 2

In this Chapter I will start by justifying why it is important to have a formal mapping between random variables defined in PR-OWL and the properties defined in OWL. The key to building the bridge that connects the deterministic ontology defined in OWL and its probabilistic extension defined in PR-OWL is to understand how to translate one to the other. On the one hand, given a concept defined in OWL, how should its uncertainty be defined in PR-OWL in a way that maintains its semantics defined in OWL? On the other hand, given a random variable defined in PR-OWL, how should it be represented in OWL in a way that respects its uncertainty already defined in PR-OWL?

In Section 4.1 I describe the need for a formal mapping between random variables defined in PR-OWL and properties defined in OWL, and I propose an approach to such a mapping. Moreover, I explain why PR-OWL 1 does not support such a mapping. Then, in Section 4.2 I present an approach to overcome the limitations in PR-OWL 1 by introducing new relationships created in PR-OWL 2. Furthermore, I present a schematic for the mapping back and forth from properties into random variables. Finally, in Section 4.3 I discuss the importance of reusing all the concepts already defined in OWL when defining a new probabilistic web ontology language (PR-OWL 2), especially data types.

Once the main ideas for mapping OWL properties to PR-OWL random variables and for using existing OWL data types in PR-OWL are presented, the main changes and characteristics of PR-OWL 2 can be presented. In Section 4.4 I show how to define a random variable in PR-OWL 2. In Section 4.5 I describe how a PR-OWL 2 reasoner must deal with entity hierarchy and polymorphism. In Section 4.6 I describe how to use the built-in RV `isA(resource, class)` in order to define type uncertainty in PR-OWL 2. In Section 4.7 I then describe the major changes in defining nodes in PR-OWL 2, which involves defining

a MEBN expression, a new concept in the language. Finally, in Section 4.8 I describe what kind of reasoning the community should expect from probabilistic ontology languages and how PR-OWL 2 is able to support them.

4.1 Why map PR-OWL Random Variables to OWL Properties?

As a running example, we consider an OWL ontology for the public procurement domain. The ontology defines concepts such as procurement, winner of a procurement, members of a committee responsible for a procurement, etc. Figure 4.1 presents an OWL ontology with a few of the concepts that would be present in this domain. In the figure we can see that a front man is defined as a person who is a front for some organization (as shown in the equivalent class expression `Person and isFrontFor some Organization` for the `FrontMan` class in Figure 4.1).

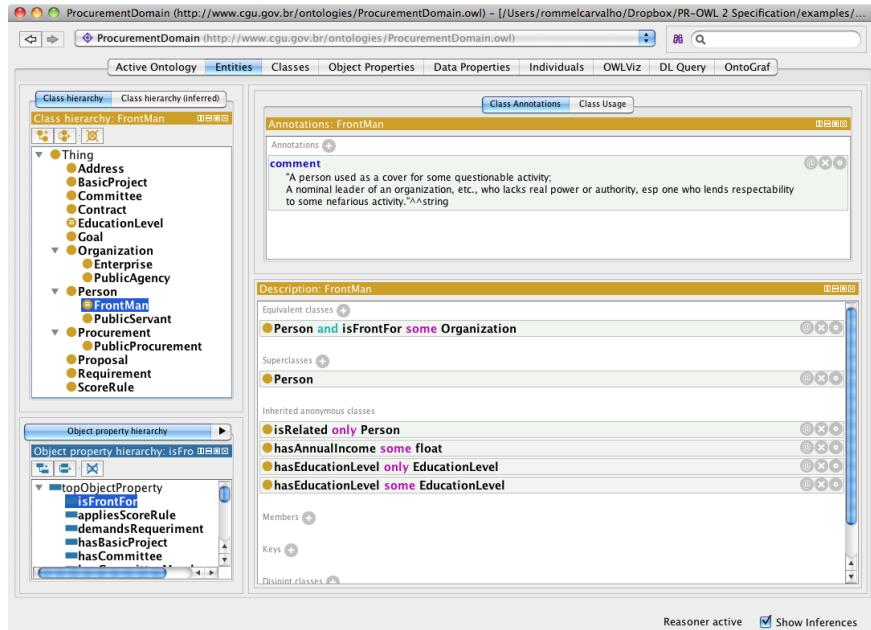


Figure 4.1: OWL ontology for the public procurement domain.

Although there is great interest in finding people acting as fronts, it is in general unknown whether a given person meets this definition. This is a typical case where we would benefit from reasoning with uncertainty. For example, if an enterprise wins a procurement for millions of dollars, but the responsible person for this enterprise makes less than 5 thousand dollars a year or if that person has only middle school education, then it is likely that this responsible person is a front for that enterprise. That is, we can identify potential fronts by examining the value of the procurement, the income of the responsible person, and his/her education level. Although we are not certain that this person is in fact a **FrontMan**, we would like to at least use the available information to draw an inference that the person is likely to be a front. This strategy is preferable to ignoring the evidence supporting this hypothesis and saying that we simply do not know whether this person is a front or not. It is also preferable to creating an arbitrary rule declaring that certain combinations of education level and income imply with certainty that a person is a **FrontMan**.

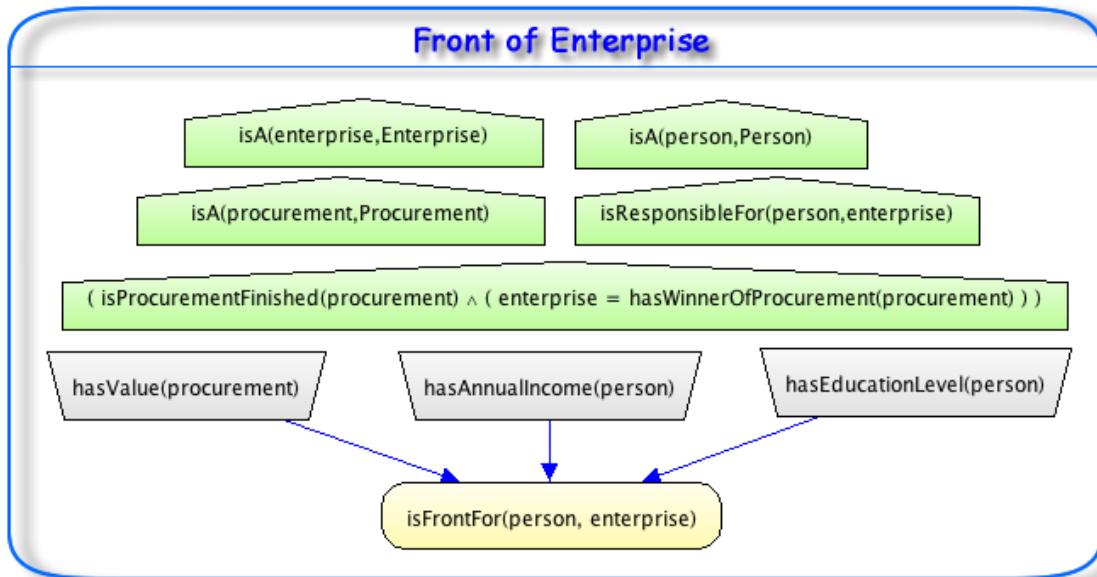


Figure 4.2: Front of an Enterprise MFragment.

This uncertain relationship is presented in Figure 4.2 as a MEBN Fragment, where we

see that the education level and annual income of a responsible person and the value of a procurement influence whether the person is front for the procurement. However, in order for the probabilistic relations described to hold, some conditions have to be satisfied, namely that the person we are considering as a possible front must be responsible for the enterprise we are examining, which is the winner of the procurement that is already finished. In other words, if the person is not responsible for the enterprise, there is no reason for this person to be considered a front for this enterprise. The same principle holds if the enterprise did not win that procurement, *i.e.*, the value of a procurement that was not won by that enterprise will not affect the likelihood of having a front for that enterprise. These conditions that must be satisfied for the probabilistic relationship to hold are depicted inside the green pentagonal shapes in the figure.

Ideally, it should be possible to use PR-OWL to reason probabilistically about uncertain aspects of an ontology based on the information already available, *i.e.*, based on existing RDF triples and knowledge that can be inferred with OWL reasoning. For instance, Figure 4.3 presents some information we could have available in an OWL ontology for the procurement domain, and uses that to generate a BN in order to draw inferences about it. So, even though we cannot say that `John_Doe` is a `FrontMan`, based on his low annual income, his low education level, and the high value of the procurement his enterprise won, we can infer that he has a high chance of being a front for that enterprise. In order to do that, we need to relate the knowledge expressed in the OWL ontology to PR-OWL random variables.

The problem with PR-OWL 1 is that it has no mapping between the random variables used in PR-OWL and the properties used in OWL. In other words, there is nothing in the language that tells us that the RV `hasEducationLevel(person)` defines the uncertainty of the OWL property `hasEducationLevel`. So, even if we have information about the education level of a specific person, for instance, if we have the triple `John_Doe hasEducationLevel middleSchool`, we would not be able to instantiate the random variable `hasEducationLevel(person)` for `John_Doe`. Although the OWL property `hasEducation`

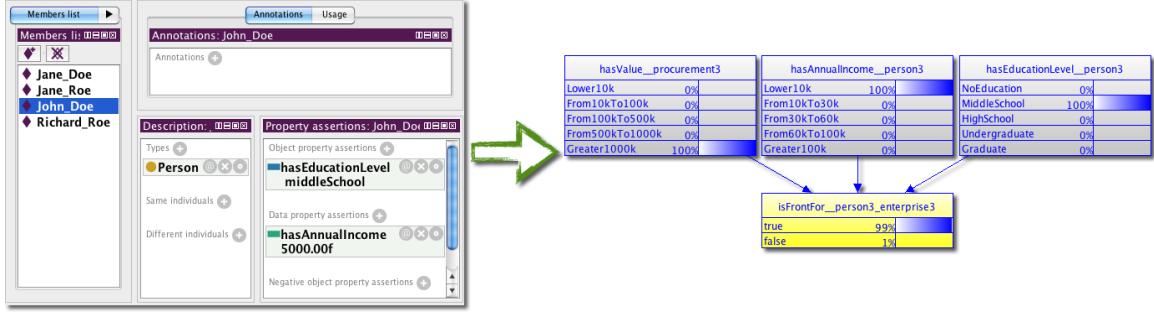


Figure 4.3: Using OWL triples for probabilistic reasoning.

and the RV `hasEducationLevel(person)` have similar syntax, there is no formal representation of this link (as depicted in Figure 4.4). In other words, we cannot use the information available in an OWL ontology (the triples with information about individuals) to perform probabilistic reasoning. Full compatibility between PR-OWL and OWL requires this ability.

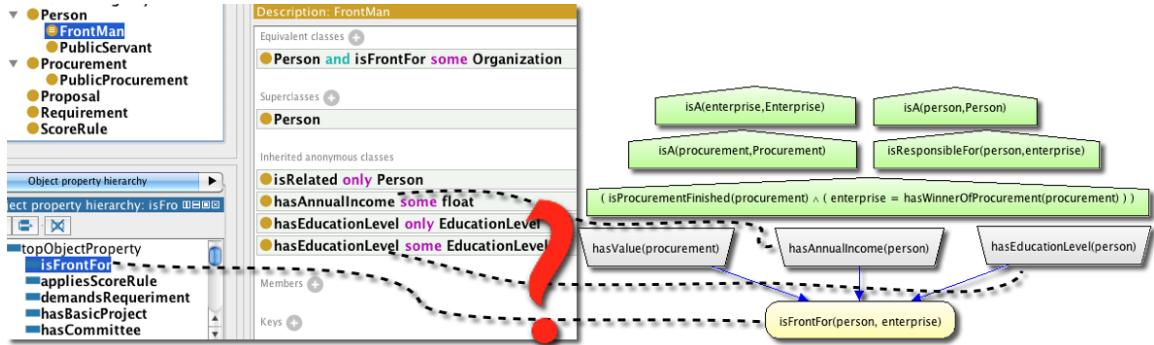


Figure 4.4: Unknown mapping between PR-OWL 1 RVs and OWL properties.

In fact, Poole *et al.* [110] states that it is not clear how to match the formalization of random variables from probabilistic theories with the concepts of individuals, classes and properties from current ontological languages like OWL. However, Poole *et al.* [110] says that “We can reconcile these views by having properties of individuals correspond to random variables.” This is exactly the approach used in this work to integrate MEBN logic and the OWL language.

4.2 The bridge joining OWL and PR-OWL

The key to building the bridge that connects the deterministic ontology defined in OWL and its probabilistic extension defined in PR-OWL is to understand how to translate one to the other. On the one hand, given a concept defined in OWL, how should its uncertainty be defined in PR-OWL in a way that maintains its semantics defined in OWL? On the other hand, given a random variable defined in PR-OWL, how should it be represented in OWL in a way that respects its uncertainty already defined in PR-OWL?

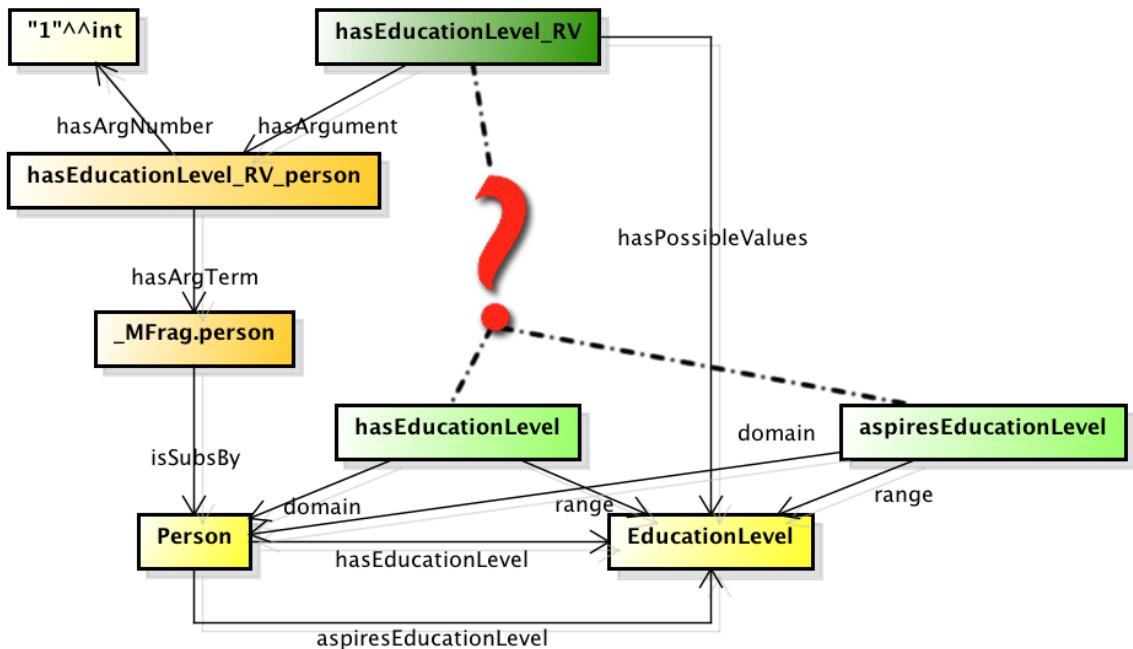


Figure 4.5: PR-OWL 1 lack of mapping from RVs to OWL properties.

Imagine we are trying to define the RV **hasEducationLevel_RV**¹, which represents the MEBN RV **hasEducationLevel(person)** used in Figure 4.2. Let's also assume that we have an OWL property called **hasEducationLevel**, which is a functional property with domain **Person** and range **EducationLevel**, and an OWL property called **aspiresEducationLevel**, which is also a functional property with domain **Person** and range **EducationLevel**. As

¹This is the OWL syntax for this RV. In MEBN we represent a RV by its name followed by the arguments in parentheses. In OWL the arguments are defined by the property **hasArgument**.

shown in Figure 4.5, in PR-OWL 1 it is not possible to distinguish whether the `hasEducationLevel_RV` is defining the uncertainty of the OWL property `hasEducationLevel` or `aspiresEducationLevel`. To clarify this problem, imagine that John Doe has only middle school (`John_Doe hasEducationLevel middleSchool`), but he aspires to have a graduate degree (`John_Doe aspiresEducationLevel graduate`). If we do not explicitly say which OWL property should be used to instantiate the `hasEducationLevel_RV`, we might end up saying that `hasEducationLevel(John_Doe) = graduate`, instead of saying that `hasEducationLevel(John_Doe) = middleSchool`, which is the intended semantics.

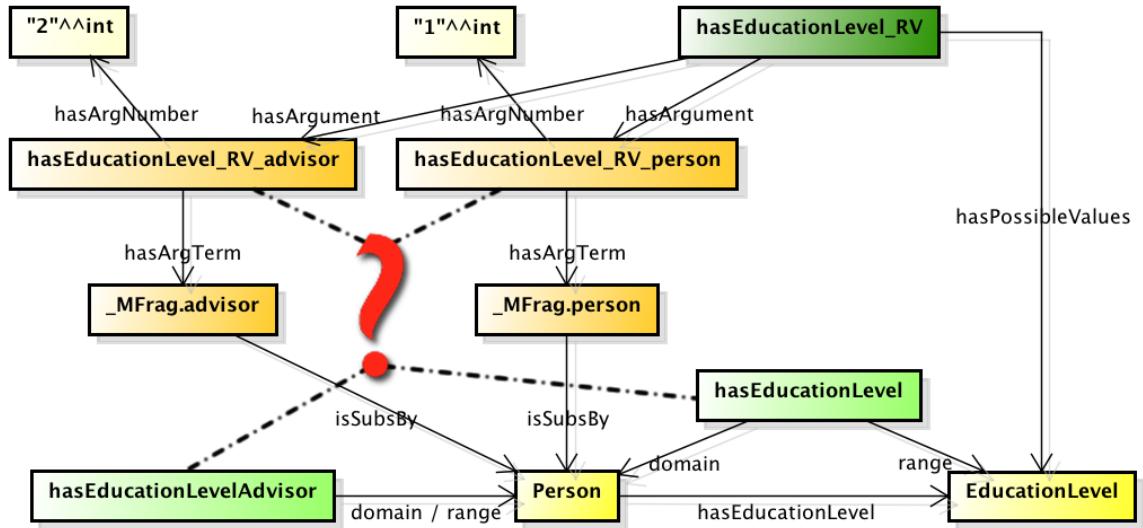


Figure 4.6: PR-OWL 1.0 lack of mapping from arguments to OWL properties.

A simple solution is to add a relation between a PR-OWL RV and the OWL property that this RV defines the uncertainty of, as suggested by Poole *et al.* [110]. In PR-OWL 2 this relation is called `definesUncertaintyOf`. However, it is not enough to have a complete mapping between RVs and OWL properties. Another problem appears when we try to define n-ary RVs. This mapping is not as straightforward as the previous one because OWL only supports binary properties (for details on suggested work arounds to define n-ary relations in OWL see [59]).

Imagine we now want to represent not only the education level a person has, but also

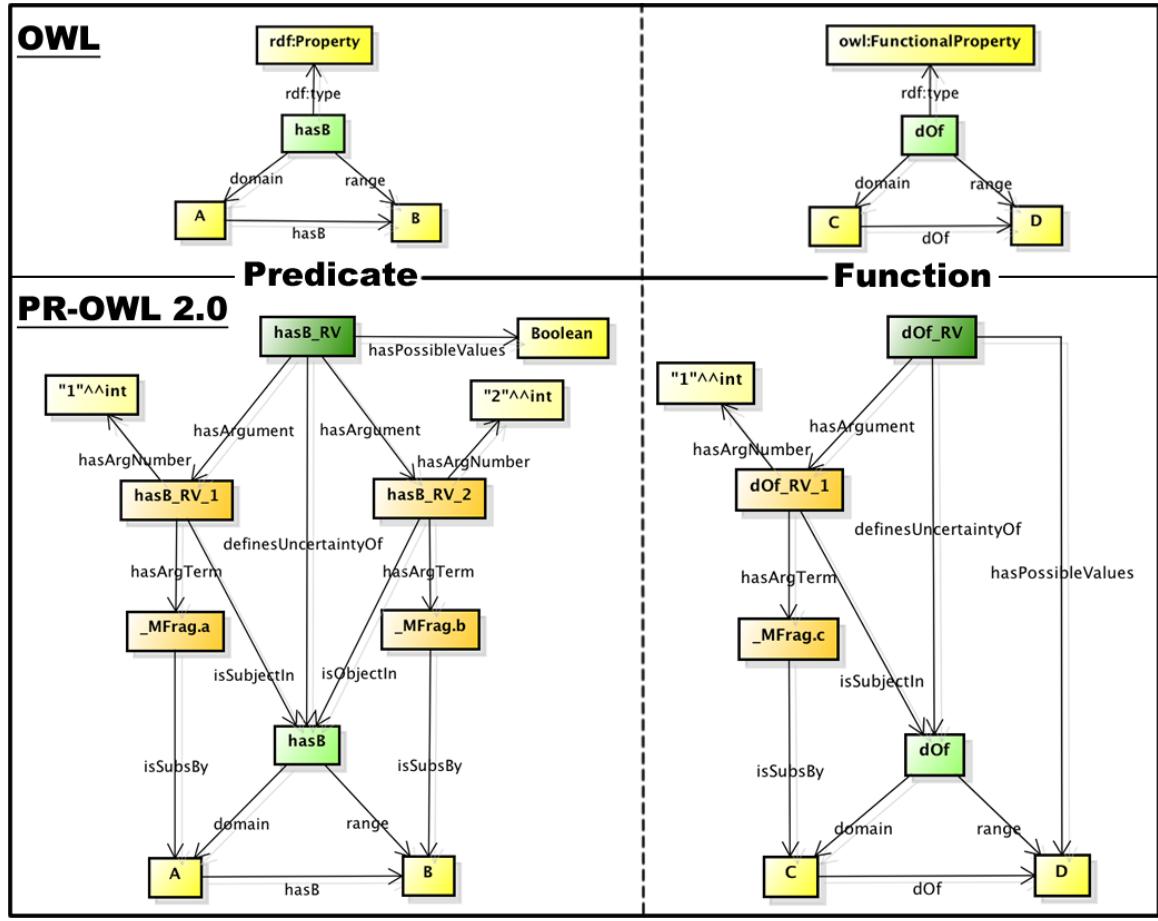


Figure 4.7: The bridge joining OWL and PR-OWL.

who was the advisor when this person attained that education level. So now, besides having the property `hasEducationLevel`, we also have the property `hasEducationLevelAdvisor`, which has `Person` as both domain and range. Thus, our RV now is `hasEducationLevel(person,advisor)`. With this new scenario, we can see that a similar problem occurs with the mapping of arguments. As it can be seen in Figure 4.6, there is nothing in PR-OWL 1 that tells which argument is associated with which property. To clarify the problem, imagine that Richard Roe has graduate education level (`Richard_Roe hasEducationLevel graduate`) and that his advisor was J. Pearl (`Richard_Roe hasEducationLevelAdvisor J_Pearl`). When instantiating the `hasEducationLevel(person,advisor)` RV, machines

would not know who is the student and who is the advisor. Although this mapping is obvious for a human being, without an explicit mapping of the arguments, machines could end up using Richard Doe as the advisor and J. Pearl as the student (`hasEducationLevel(J_Pearl, Richard_Roe)`), instead of using J. Pearl as the advisor and Richard Doe as the student (`hasEducationLevel(Richard_Roe, J_Pearl)`).

As expected, to a similar problem we apply a similar solution. In PR-OWL 2 we have a relation between an argument to a RV and the OWL property it refers to. However, unlike the RV mapping, the argument mapping refers to either the domain or the range of a property, not to the property itself. For instance, in the `hasEducationLevel(person, advisor)` RV, the `person` argument refers to the domain of the OWL property `hasEducationLevel`, which is a `Person`. The `advisor` argument, on the other hand, refers to the range of the OWL property `hasEducationLevelAdvisor`, which is also a `Person`, but a different one (a person cannot be his/her own advisor). Therefore, in order to differentiate when the argument refers to the domain or to the range of a property, we add to PR-OWL 2 the relations `isSubjectIn` and `isObjectIn`.

More examples of random variables in this new format can be found in [21]. Here, a schematic is given in Figure 4.7 for the 2-way mapping between triples and random variables. Functions and predicates are considered as separate cases.

If a property (`hasB` or `dOf`) is defined in OWL, then its domain and range are already represented (`A` and `B`; `C` and `D`, respectively). The first thing to be done is to create the corresponding RV in PR-OWL (`hasB_RV` and `dOf_RV`, respectively) and link it to this OWL property through the property `definesUncertaintyOf`.

For binary relations, the domain of the property (`A` and `C`, respectively) will usually be the type (`isSubsBy`) of the variable (`_Mfrag.a` and `_Mfrag.c`, respectively) used in the first argument (`hasB_RV_1` and `dOf_RV_1`, respectively) of the RV. For n-ary relations see example given earlier in this Section on the RV `hasEducationalLevel(person, advisor)` and also [21].

If the property is non-functional (`hasB`), then it represents a predicate that may be true

or false. Thus, instead of having the possible values of the RV in PR-OWL (`hasB_RV`) being the range of the OWL property (`B`), it must be `Boolean`. So, its range (`B`) has to be mapped to the second argument (`hasB_RV_2`) of the RV, the same way the domain (`A`) was mapped to the first argument (`hasB_RV_1`) of the RV. On the other hand, if the the property is functional (`dOf`), the possible values of its RV (`dOf_RV`) must be the same as its range (`D`).

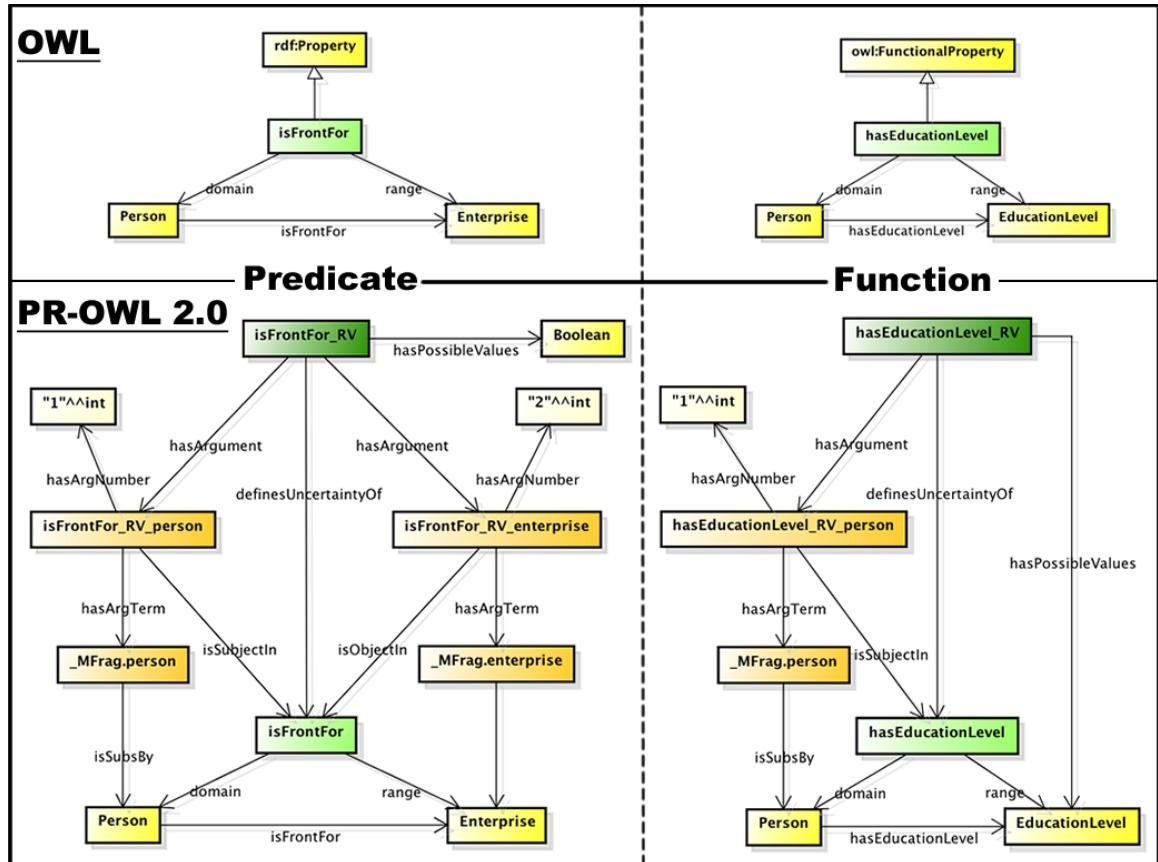


Figure 4.8: Example of binary RVs mapping to OWL properties for both predicate and function.

It is important to note that not only is the RV linked to the OWL property by `defineUncertaintyOf`, but also its arguments are linked to their respective OWL properties by either `isSubjectIn` or `isObjectIn`, depending on what they refer to (domain or range of the OWL property, respectively). This feature is especially important when dealing with n-ary relations, where each variable will be associated with a different OWL property (see

explanation of Figure 4.6 earlier in this Section for details).

Finally, if the RV is already defined in PR-OWL with all its arguments and its possible values, the only thing that needs to be done is to create the corresponding OWL property, link the RV to it using `definesUncertaintyOf`, create the OWL properties for the arguments, if necessary, link them using either `isSubjectIn` or `isObjectIn`, depending on what they refer to (domain or range of the OWL property, respectively), and make sure that the domain and range of the property matches the RV definition, as explained previously.

Figure 4.8 presents examples of instantiations of the schematic just presented. In it we have the mapping of the RV `isFrontFor(person,enterprise)` to the OWL property `isFront`, which is a predicate, and the mapping of the RV `hasEducationLevel(person)` to the OWL property `hasEducationLevel`, which is a function.

The mapping described in this Section provides the basis for a formal definition of consistency between a PR-OWL probabilistic ontology and an OWL ontology, in which rules in the OWL ontology correspond to probability one assertions in the PR-OWL ontology. A formal notion of consistency can lead to development of consistency checking algorithms.

4.3 Extending PR-OWL to Use OWL's Types

One of the main concerns when developing OWL [66] was to keep the same semantics of its predecessors, RDF and XML, which meant reusing all the concepts already defined in those languages, including primitive types, such as string, Boolean, decimal, etc. However, PR-OWL 1 does not make use of the primitive types used in OWL. For instance, PR-OWL 1 defines *Boolean* as an individual of the class *MetaEntity*, as shown in Figure 4.9, but does not keep any relation to the Boolean type used in OWL.

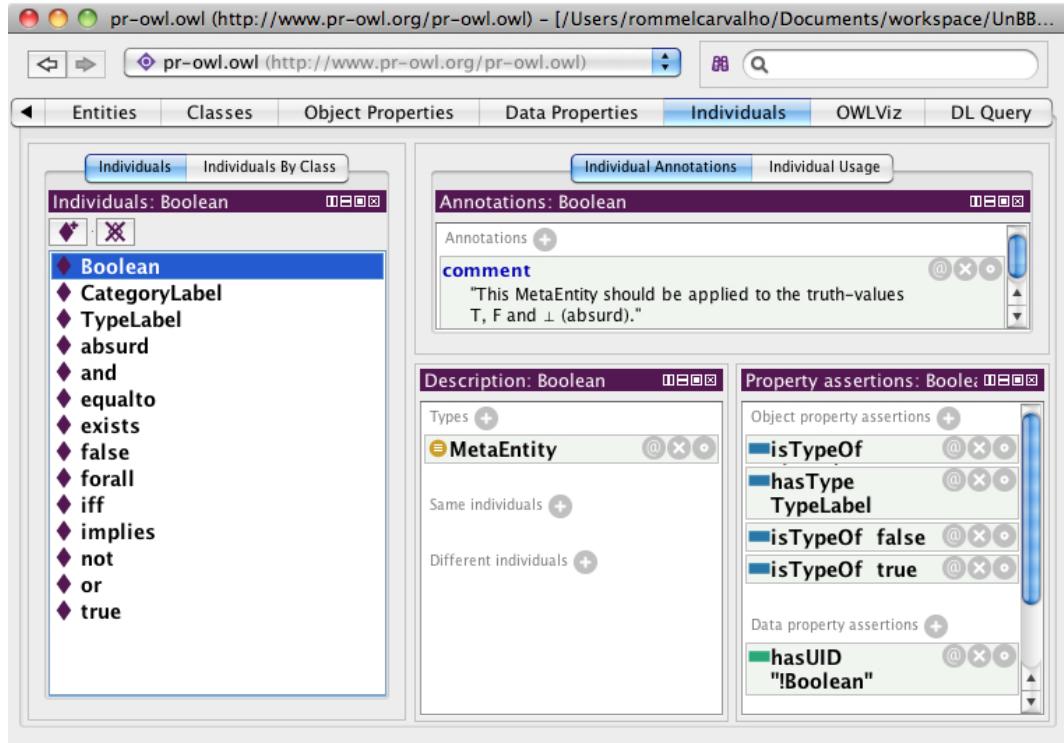


Figure 4.9: Boolean individual defined in PR-OWL.

If we wanted to define a continuous random variable for the annual income of a person in PR-OWL, we would need to define the real numbers, even though they are already defined in OWL. Moreover, concepts that use this primitive type in OWL would not be understood in PR-OWL, in other words, they lack compatibility as far as primitive types are concerned.

Figure 4.10 shows the different types of entities defined in PR-OWL. A possible approach to keep OWL's semantics is to avoid defining new types of entities and use what is already available in OWL. For instance, the class *ObjectEntity* can be substituted by the OWL class *Thing*: after all, according to [27] *ObjectEntity* aggregates the MEBN entities that are real world concepts of interest in a domain. They are akin to objects in Object-Oriented (OO) models and to frames in frame-based knowledge systems. In other words, they can be replaced by OWL classes, although these classes will have to respect additional semantics from PR-OWL.

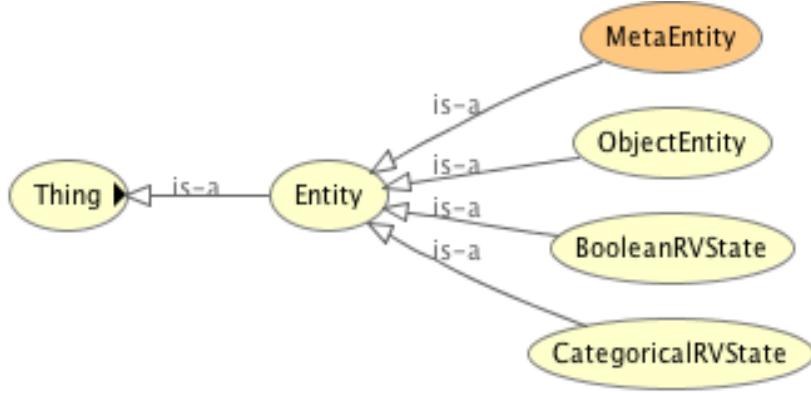


Figure 4.10: The different types of entities defined in PR-OWL.

According to [27] *CategoricalRVState* is used to represent a list of mutually exclusive, collectively exhaustive states, which in turn are possible states of random variables, represented by nodes in PR-OWL. Therefore, it can be replaced by *DataOneOf* if it needs to enumerate data types or *ObjectOneOf* if it needs to enumerate objects. These concepts allow the enumeration of literals and individuals, respectively (see [97] for more details).

BooleanRVState can be replaced by the Boolean data type present in OWL. Finally, the *MetaEntity* class, which includes all the entities that convey specific definitions about entities (*e.g.* `typelabels` that name the possible types of entities), can be eliminated since all other entities were replaced by a concept already present in OWL.

4.4 Defining a Random Variable in PR-OWL 2

Now that the main ideas for mapping OWL properties to PR-OWL random variables and for using existing OWL data types in PR-OWL have been presented, it can be shown how to define a random variable in PR-OWL 2.

The first change to notice, is that in PR-OWL 2 there is a separate class to define random variables, and nodes make reference to this class. In PR-OWL 1, there is no separate random class. A random variable is identified with the resident node in its home Mfrag, and input and context nodes make reference to the resident node itself.

Besides the clear difference in semantics, *i.e.*, a resident node is not the same thing as a random variable, there are other advantages in making such distinction.

First, by having input and context nodes point to the random variable instead of pointing to a specific resident node, it is possible to decide dynamically which distribution to use. This is a form of polymorphism that can be very useful to the modeler. For instance, it is possible to define a distribution for the random variable `RV.hasHeight` given the domain is a `Person` and another distribution given the domain is a `VolleyballPlayer` (subclass of `Person`). So, at runtime, if someone is known only to be a person, then the first distribution will be used, however, if it is known that this person is in fact a volleyball player, then the second and more specific distribution will be used. In PR-OWL 1, since the input and context nodes were linked directly to a resident node, the user would be forced to choose one of the two distributions before runtime, during modeling. This was clearly not flexible enough to support polymorphism.

Second, in MEBN, when at least one of the context nodes is not valid, the distribution defined for its resident nodes are not correct and a default distribution should be used instead. This default distribution should be independent of the MFragment. In other words, there should be a unique default distribution for a random variable, independent of context. In PR-OWL 1, every resident node had its own default distribution. This could eventually lead to inconsistencies. Imagine there are two different distributions for the same random variable and that each one has a different default distribution. Now, imagine that in runtime none of these MFrags could be instantiated (invalid context nodes). Which default distribution should be used? It is clear that there is no right answer, which could lead to different answers depending on the choice. This problem does not happen in PR-OWL 2, since the default distribution is defined only once for the random variable itself, and is independent of context (it applies in any situation).

Third, in PR-OWL 2 the mapping between a PR-OWL random variable and an OWL property is defined only once. In PR-OWL 1, the mapping would have to be defined for every resident node that maps to that OWL property. Besides being repetitive and unnecessary,

it is easy to see that this could eventually lead to mapping inconsistencies.

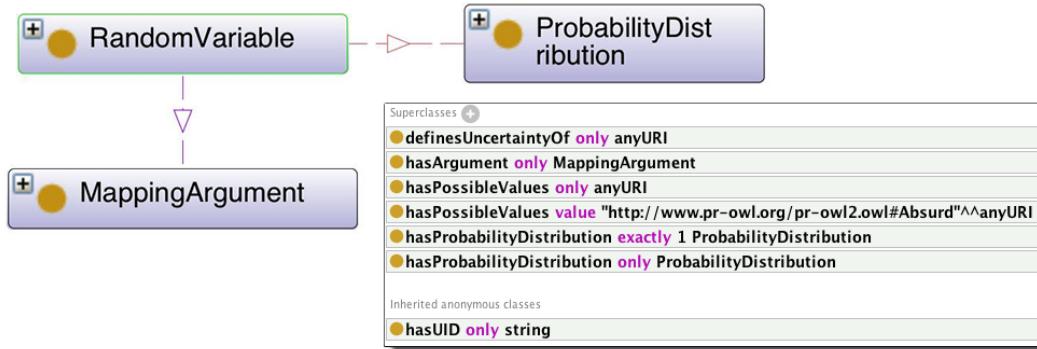


Figure 4.11: The OWL restrictions of the `RandomVariable` class.

In PR-OWL 2 a random variable defines the uncertainty of the outcome related to a specific property, which has its semantics defined in OWL. There are four main concepts that need to be defined for every random variable (see Figure 4.11)²: a link to the OWL property it defines the uncertainty of (represented by the property `prowl2:definesUncertaintyOf`); the domain of the random variable defined by its arguments (represented by the property `prowl2:hasArgument`); the range or possible outcomes of the random variable (represented by the property `prowl2:hasPossibleValues`); and finally, its distribution (represented by the property `prowl2:hasProbabilityDistribution`). Each of these concepts will be discussed in more detail in the following subsections before presenting examples of random variables.

In MEBN, every random variable has absurd as one of its possible values. Therefore, in PR-OWL this is also the case.

²This graph was generated using the OntoGraf plugin for Protégé, which comes with the default distribution of Protégé 4. The homepage of the OntoGraf project is <http://protegewiki.stanford.edu/wiki/OntoGraf>.

4.4.1 Mutually Exclusive and Collectively Exhaustive Outcomes

A random variable is a function that maps elements of a sample space to elements of an outcome set. The sample space represents all possible outcomes of an event or experiment. The actual outcome is unknown before the event happens. Likelihoods of the different possible outcomes are represented by a probability distribution. In statistics, the outcome set is usually taken to be the real numbers. This real number can represent categorical values (*e.g.*, low, medium, high), Boolean values (true and false), or some other set of mutually exclusive and collectively exhaustive outcomes. In the Artificial Intelligence (AI) literature, random variables are typically allowed to range over values in an arbitrary set. This convention makes it much more straightforward to represent semantics.

As explained before, in PR-OWL 2, the possible values or outcomes of a random variable are instances of classes and data types. When specifying that a random variable will have individuals of a class as its possible outcomes, it is reasonable to assume that all known individuals of that class form a set of collectively exhaustive outcomes. However, the assumptions about individuals in OWL are not enough to guarantee these individuals are mutually exclusive. More specifically, although OWL provides a way to express unique names, it also allows two different names to point to the same object in the real world. For instance, assume the random variable `hasMother(person)` has the individuals of class `Person` as its possible outcomes. Suppose amongst the individuals of class `Person`, we have `Rebecca` and `Becky`, but we happen to know that `Rebecca` is the same individual as `Becky` (supported by the OWL property `sameAs`). Therefore, if we simply list all known instances of `Person` as possible values of `hasMother(person)`, we would have at least two states meaning the same thing, *i.e.*, the outcomes would not be mutually exclusive.

To overcome this shortcoming, PR-OWL 2 follows the MEBN and PR-OWL 1 convention, and assumes that every individual has a unique ID associated to it. For instance, both `Rebecca` and `Becky` from our previous example would have the same unique ID (*e.g.*, ‘‘`Rebecca`’’). This is represented by the data property `hasUID`, which has `Thing` as its domain and `string` as its range. This is actually the same solution applied in PR-OWL 1,

however, in PR-OWL 1 the domain of this property was `Entity`, since PR-OWL 1 had a special set of classes to represent entities and types.

The uniqueness of the ID is guaranteed by the OWL restriction on the `hasUID` property that says it is a functional property. The only rules that have to be guaranteed by the PR-OWL reasoner are:

1. All individuals that are the same (related by the `sameAs` property), must have the same unique ID (same string for the property `hasUID`);
2. All individuals that are not the same must have different unique IDs.

This is not the only problem of having different individuals meaning the same thing. Suppose we have a finding saying that `hasMother(Josh) = Rebecca`, and that `Josh sameAs Joshua` (*i.e.*, they have the same unique ID). If we ask the PR-OWL reasoner who is the mother of `Joshua` by replacing the argument person from the `hasMother(person)` with `Joshua`, the PR-OWL reasoner might not be able to understand that `hasMother(Joshua) = Rebecca`, as expected. To avoid such problem, the unique ID should be used not only for possible outcomes, but also for arguments of random variables. This way we would have a unique representation for the finding and no matter what individual we use in our query, as long as the reasoner uses its unique ID, the answer would always be the same and correct. In fact, this is required by MEBN for defining RVs. In order to define RVs with arguments that are not UIDs, MEBN uses built-in MFrags. In other words, non UIDs can be used as arguments (*e.g.*, individuals), as long as the PR-OWL 2 reasoner replaces them with their UID to avoid the problem described.

4.4.2 Avoiding OWL Full

The careful reader might have noticed that although we have directly mapped random variables to properties in Figure 4.7, in Figure 4.11 this mapping is slightly different. In order to properly say that a random variable defines the uncertainty of some property, we would have to define the class `RandomVariable` having the restriction `definesUncertaintyOf`

`some rdf:Property`. However, using such a restriction would require OWL Full, since according to [53], “IRIs from the reserved vocabulary other than owl:Thing and owl:Nothing must not be used to identify classes in an OWL 2 DL ontology.” In order to keep PR-OWL 2 as an OWL 2 DL ontology, we avoided this restriction and replaced it by `definesUncertaintyOf some xsd:anyURI`. Otherwise, we would not be able to take advantage of the reasoners available for OWL, since they are usually not capable of reasoning with OWL Full ontologies.

Note that there are semantics that OWL-DL reasoners cannot capture, but PR-OWL reasoners are expected to respect. In other words, even though the property `definesUncertaintyOf` accepts any URI, the PR-OWL reasoner has to make sure the URI is in fact a property. That is, PR-OWL 2 is a proper extension of OWL-DL, having semantic restrictions beyond those represented in the OWL ontology. PR-OWL reasoners must respect the extended semantics.

This solution of using URIs is used more than once in PR-OWL 2 as a means to capture extended semantics while still allowing the use of OWL-DL reasoners to reason about aspects of PR-OWL ontologies that OWL-DL can represent. Another example is the property `hasPossibleValues`, which is used to define the possible values of a random variable. Although it accepts any URI (by the restriction `hasPossibleValues only anyURI`), the semantics of PR-OWL defines that the only possible URIs are those that point to a class or a data type. This property only defines the type of the possible values, however, the possible values will be the UIDs of the individuals of the allowed type (individuals that represent the same object will only be represented once by their UID).

4.4.3 Built-in Random Variables

Figure 4.12 presents not only the hierarchy but also the built-in random variables defined in PR-OWL 2. Although it is not explicitly shown, there is a “soft” link between `RandomVariable` and `Absurd`, represented by the restriction `hasPossibleValues value`

‘‘<http://www.pr-owl.org/pr-owl2.owl#Absurd>’’ $\wedge\wedge$ anyURI (see Figure 4.11).³

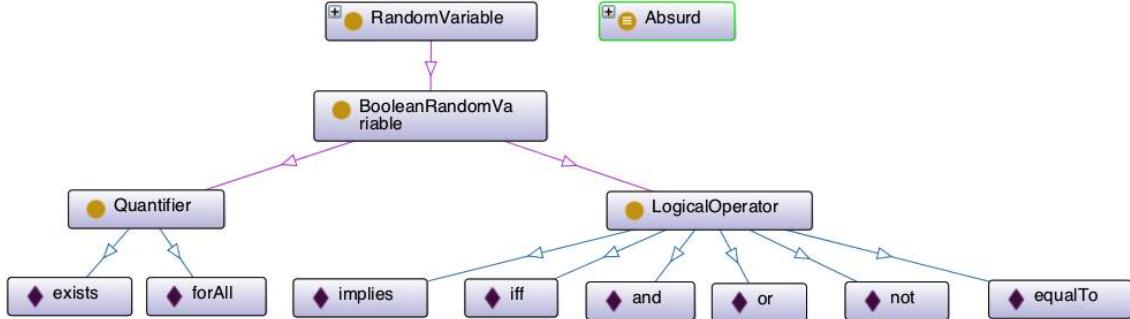


Figure 4.12: Graph with the main concepts for defining random variables.

A BooleanRandomVariable is a special kind of RandomVariable, which represents random variables that can only have Boolean values as their possible values or range, guaranteed by the restriction hasPossibleValues value ‘‘<http://www.w3.org/2001/XMLSchema#boolean>’’ $\wedge\wedge$ anyURI (see Figure 4.13).

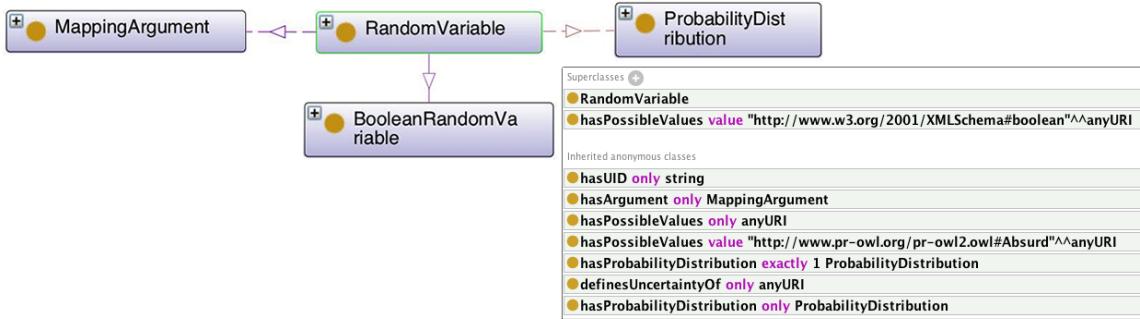


Figure 4.13: The OWL restrictions of the BooleanRandomVariable class.

³We call this link “soft” because we are using the data type URI to make reference to the Absurd class. Usually the link would be made to the class Absurd directly, through the use of an object property, *i.e.*, the range of the property would be a class and not the data type URI. As explained before, this was done in order to allow the use of OWL DL reasoners. The link is not shown in Figure 4.12 exactly because this link is associated to the URI (not shown in the Figure), and not with the class Absurd itself, which is shown in the Figure.

A `LogicalOperator` is a special kind of `BooleanRandomVariable`, which represents first-order logic (FOL) operators. These logic operator random variables are mostly used to express FOL formulas using MEBN expressions (see Subsection 4.7.3 for details on how to define FOL formulas). Figure 4.14 presents the OWL restrictions for this class.

Since these operators can represent more expressive formulas than those represented in OWL DL, there is no explicit mapping of these RVs to OWL properties. This is a special case for these built-in random variables. Nevertheless, they can be mapped (in a future version) to an OWL ontology that defines FOL logic operators.

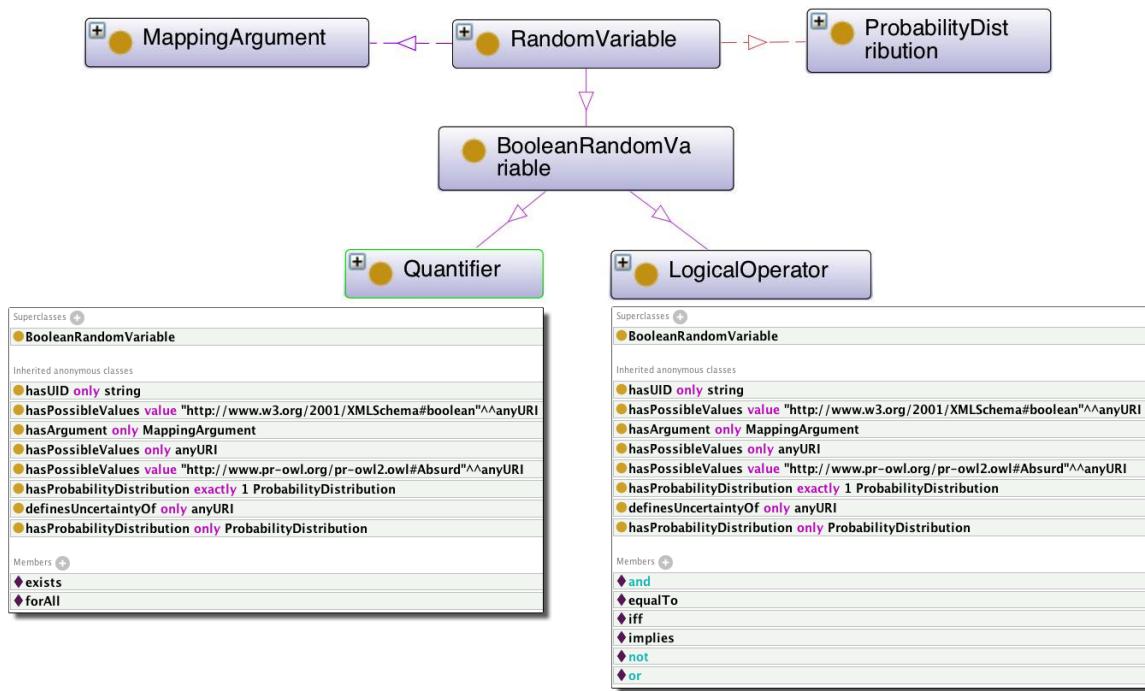


Figure 4.14: The OWL restrictions of the classes `LogicalOperator` and `Quantifier`.

The built-in logical operators available in PR-OWL 2 are the same and also represented as instances as in PR-OWL 1. Namely they are:

`and` represents the FOL 'and' operator and must have two arguments;

`or` represents the FOL 'or' operator and must have two arguments;

`not` represents the FOL 'not' operator and must have one argument;

`equalTo` represents the FOL ' $=$ ' operator and must have two arguments;

`implies` represents the FOL ' \Rightarrow ' operator, which is an 'if then' statement, and must have two arguments;

`iff` represents the FOL ' \Leftrightarrow ' operator, which is an 'if and only if' statement, and must have two arguments.

A **Quantifier** is a special kind of `BooleanRandomVariable`, which represents first-order (FOL) quantifiers. These quantifier RVs are mostly used to express FOL formulas using MEBN expressions (see Subsection 4.7.3 for details on how to define FOL formulas). Figure 4.14 presents the OWL restrictions for this class.

Since these quantifiers can represent more expressive formulas than those represented in OWL DL, there is no explicit mapping of these RVs to OWL properties. This is a special case for these built-in random variables. Nevertheless, they can be mapped (in a future version) to an OWL ontology that defines FOL quantifiers.

The built-in quantifiers available in PR-OWL 2 are the same and also represented as instances as in PR-OWL 1. Namely they are:

`exists` represents the FOL ' \exists ' quantifier and must have two arguments (one exemplar and one formula which it is quantifying over). If more than one variable needs to be quantified over, *i.e.*, if you need more than one exemplar on the same formula, it is necessary to nest this quantifier with another quantifier over the other bound variable. This restriction is to be consistent with the MEBN specification presented in [78];

`forall` represents the FOL ' \forall ' quantifier and must have two arguments (one exemplar and one formula which it is quantifying over). If more than one variable needs to be quantified over, *i.e.*, if you need more than one exemplar on the same formula, it is necessary to nest this quantifier with another quantifier over the other bound variable. This restriction is to be consistent with the MEBN specification presented in [78].

4.4.4 Defining Arguments for Random Variables

Besides mapping a random variable to an OWL property and defining its possible values, a random variable needs to define the arguments it has, which basically represent the domain of the RV.

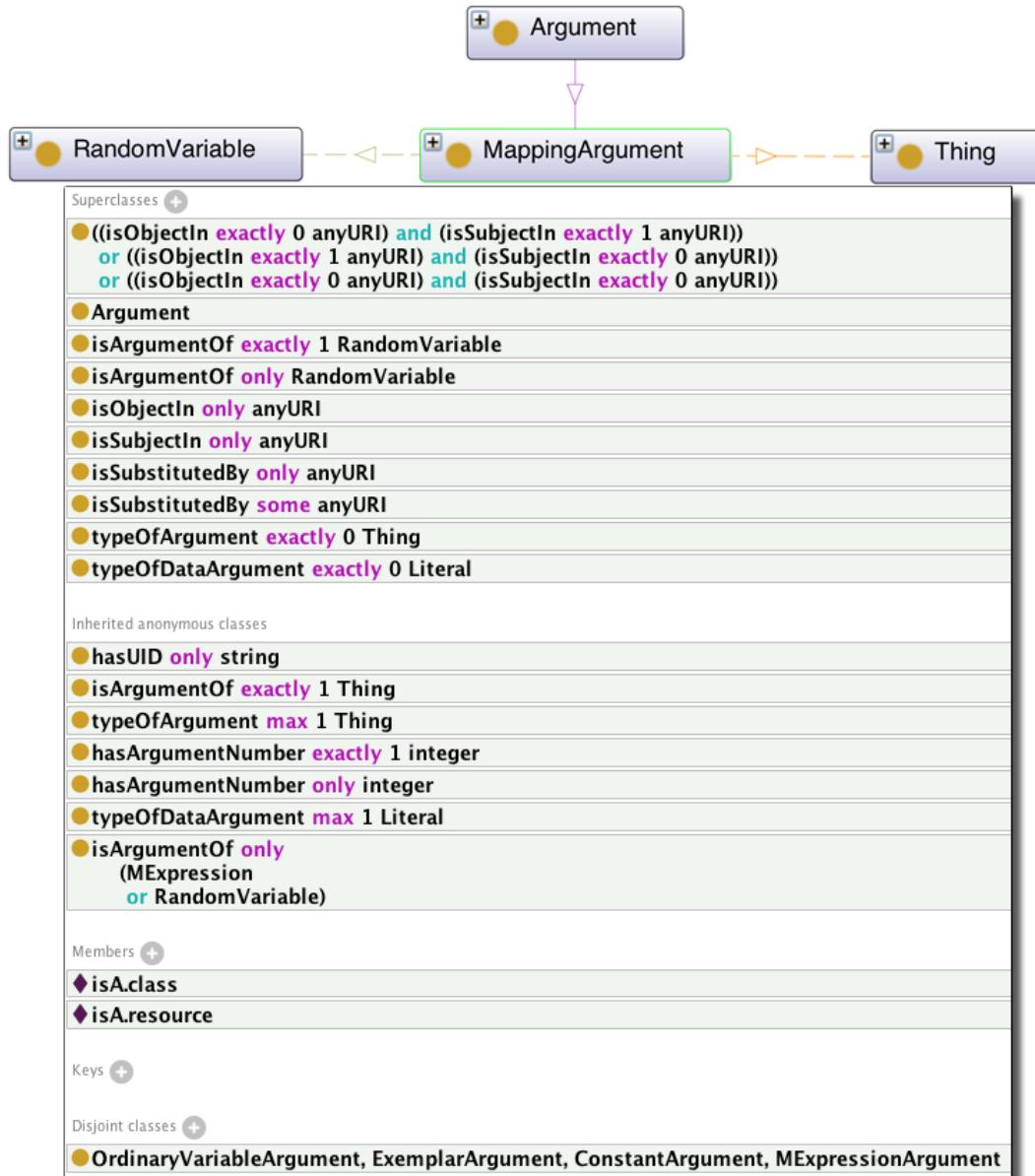


Figure 4.15: The OWL restrictions of the `MappingArgument` class.

`MappingArgument` is used to map random variable arguments to OWL properties domain or range. On the one hand, to map a random variable argument to a domain of some OWL property, it is necessary to say that this argument `isSubjectIn` some OWL property. On the other hand, to map a random variable argument to a range of some OWL property, it is necessary to say that this argument `isObjectIn` some OWL property. In both cases, the type of the argument has to be consistent with the OWL property it points to. If `isSubjectIn` is used, then the type of the argument (defined by the property `isSubstitutedBy`) has to be the same as the domain of the property it points to. However, if `isObjectIn` is used instead, then the type of the argument (defined by the property `isSubstitutedBy`) has to be the same as the range of the property it points to.

Since OWL DL does not allow the use of restricted vocabulary when defining the ontology, instead of saying that `isSubjectIn` only `rdf:Property` and `isObjectIn` only `rdf:Property`, the restrictions are defined as `isSubjectIn` only `rdfs:anyURI` and `isObjectIn` only `rdfs:anyURI` (see Section 4.4.2 for more information). Nevertheless, the semantics of PR-OWL enforce that these URIs have to point to RDF properties. The same reasoning applies to the restriction `isSubstitutedBy` only/some `rdfs:anyURI`. The semantics of PR-OWL enforce that these URIs have to point to either classes or data types. Figure 4.15 presents the OWL restrictions for this class.

4.4.5 Defining Distributions for Random Variables

Finally, once the random variable, its arguments, possible values, and respective mappings have been defined, it is necessary to define its default probability distributions. The distribution defined here will be used if there is no other option, *i.e.*, if there is no MFragment with a resident node pointing to this random variable or if there is but none has its context nodes satisfied.

The `ProbabilityDistribution` class is used to define the local distributions for each resident node (these local distributions apply only if all context nodes in the MFragment are satisfied), and the default distribution for a random variable (to be used if none of the

local distributions in any of its home MFrags applies). A probability distribution can be described using an application dependent declarative format, such as a UnBBayes local probability distribution (LPD), or via a PR-OWL table (which has probability assignments as its cells). Figure 4.16 presents the OWL restrictions for this class.

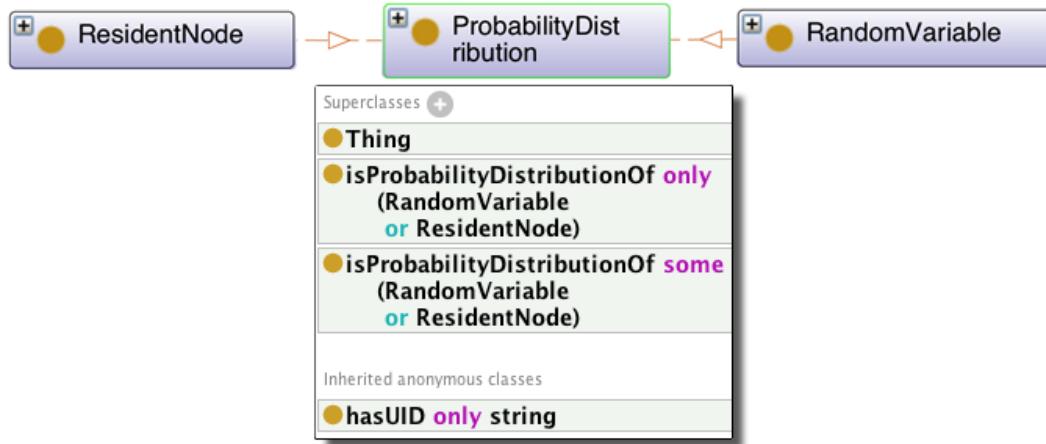


Figure 4.16: The OWL restrictions of the `ProbabilityDistribution` class.

Figure 4.17 presents the main concepts and their relations for defining probability distributions. It can be seen that both `ResidentNode` and `RandomVariable` have probability distributions. There are two types of probability distributions, `PR-OWLTable` and `DeclarativeDistribution`. A `PR-OWLTable` has probability assignments (`ProbabilityAssignment`), which depend on conditioning states from the parents (`ConditioningState`). A `DeclarativeDistribution` is defined by some script, which follows some application-specific grammar.

For detailed information on how to define probability distributions, see Appendix A Section A.2.

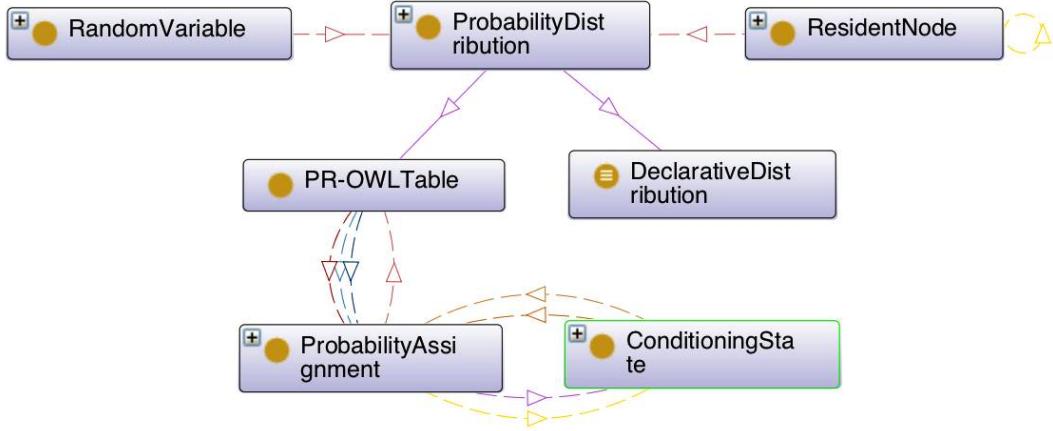


Figure 4.17: Graph with the main concepts and their relations for defining probabilistic distributions.

4.4.6 Examples of Random Variables

In the following subsections we will present different examples of random variables. First we present the Boolean random variable `isRelated(person1, person2)`, which represents whether two people are related. Then we present an example of an object random variable, `livesAt(person)`, which represents where a given person lives. Finally we present an example of a data random variable, `hasAnnualIncome(person)`, which represents how much a person makes a year.

It is worth noting that until PR-OWL 2 it was not possible to define random variables with data types as their possible values (except the built-in Boolean data type defined in PR-OWL 1). This type of construction is one of the contributions of this work, due to the use of existing types supported by OWL (*e.g.*, float, date, time, int, etc).

Boolean Random Variable

Imagine we want to represent uncertainty about whether two people are related. Suppose that there is already an OWL property that has that semantics (`Person isRelated Person`). Now, in order to define uncertainty about this property we need to create a random variable and map it to this OWL property. Since this property is a predicate we need to create a Boolean random variable (`RV.isRelated`), which defines the uncertainty of the OWL property `isRelated`. Listing 4.1 presents this random variable. It has two arguments, `RV.isRelated.MA.person1` and `RV.isRelated.MA.person2`, its probability distribution is represented by `ex:RV.isRelated.PT.dist1`, and its range is Boolean (inherited from its type `BooleanRandomVariable`).

Listing 4.1: Definition of the Boolean random variable `isRelated(person1, person2)`

```
1 Individual: RV.isRelated
2   Types:
3     pr-owl2:BooleanRandomVariable
4   Facts:
5     pr-owl2:hasArgument  RV.isRelated.MA.person1 ,
6     pr-owl2:hasArgument  RV.isRelated.MA.person2 ,
7     pr-owl2:hasProbabilityDistribution  RV.isRelated.PT.dist1 ,
8     pr-owl2:definesUncertaintyOf  "&ex; isRelated"^^xsd:anyURI
```

Until now we have just given names but no semantics to the arguments and probability distribution of this random variable.

Listing 4.2 presents the semantics for the two arguments. Argument `RV.isRelated.MA.person1` is the first argument of the random variable `RV.isRelated` and it can be substituted by the class `Person`. This argument is the subject in the OWL property `isRelated`. Argument `RV.isRelated.MA.person2` is the second argument of the random variable `RV.isRelated` and it can be substituted by the class `Person`. This argument is the object in the OWL property `isRelated`.

Listing 4.2: Arguments of the Boolean random variable `isRelated(person1, person2)`

```

1 Individual: RV.isRelated.MA.person1
2   Types:
3     pr-owl2:MappingArgument
4   Facts:
5     pr-owl2:isArgumentOf RV.isRelated ,
6     pr-owl2:hasArgumentNumber 1,
7     pr-owl2:isSubstitutedBy "&ex; Person"^^xsd:anyURI ,
8     pr-owl2:isSubjectIn "&ex; isRelated"^^xsd:anyURI
9
10 Individual: RV.isRelated.MA.person2
11   Types:
12     pr-owl2:MappingArgument
13   Facts:
14     pr-owl2:hasArgumentNumber 2,
15     pr-owl2:isArgumentOf RV.isRelated ,
16     pr-owl2:isSubstitutedBy "&ex; Person"^^xsd:anyURI ,
17     pr-owl2:isObjectIn "&ex; isRelated"^^xsd:anyURI

```

Table 4.1: Table representing the distribution for the random variable `isRelated(person1, person2)`.

State	Probability
True	0.001
False	0.999
Absurd	0.0

Table 4.1⁴ presents the probability distribution for the random variable `RV.isRelated`. Listing 4.3 presents how the distribution depicted in Table 4.1 is represented as a PR-OWL table. As seen in Table 4.1, since the random variable has 3 states and no parents, the number of probability assignments is 3. The assignment `RV.isRelated.PT.dist1.PA.assign1` assigns probability .001 to the state `true`. The assignment `RV.isRelated.PT.dist1.PA.assign2` assigns probability .999 to the state `false`.

⁴This table follows UnBBayes default way of representing probability distributions, where the states of the child are represented in each row and the combination of parent states are represented on the header of the table. In UnBBayes the rows have to sum to 1.

Finally, the assignment `RV.isRelated.PT.dist1.PA.assign3` assigns probability 0 to the state `absurd`.

Listing 4.3: PR-OWL table for the random variable `isRelated(person1, person2)`

```

1 Individual: RV.isRelated.PT.dist1
2   Types:
3     pr-owl2:PR-OWLTable
4   Facts:
5     pr-owl2:isProbabilityDistributionOf  RV.isRelated ,
6     pr-owl2:hasProbabilityAssignment  RV.isRelated.PT.dist1.PA.assign1 ,
7     pr-owl2:hasProbabilityAssignment  RV.isRelated.PT.dist1.PA.assign2 ,
8     pr-owl2:hasProbabilityAssignment  RV.isRelated.PT.dist1.PA.assign3
9
10 Individual: RV.isRelated.PT.dist1.PA.assign1
11   Types:
12     pr-owl2:ProbabilityAssignment
13   Facts:
14     pr-owl2:isProbabilityAssignmentIn  RV.isRelated.PT.dist1 ,
15     pr-owl2:hasStateName  "true"^^xsd:string ,
16     pr-owl2:hasStateProbability  .001f
17
18 Individual: RV.isRelated.PT.dist1.PA.assign2
19   Types:
20     pr-owl2:ProbabilityAssignment
21   Facts:
22     pr-owl2:isProbabilityAssignmentIn  RV.isRelated.PT.dist1 ,
23     pr-owl2:hasStateName  "false"^^xsd:string ,
24     pr-owl2:hasStateProbability  .999f
25
26 Individual: RV.isRelated.PT.dist1.PA.assign3
27   Types:
28     pr-owl2:ProbabilityAssignment
29   Facts:
30     pr-owl2:isProbabilityAssignmentIn  RV.isRelated.PT.dist1 ,
31     pr-owl2:hasStateName  "absurd"^^xsd:string ,
32     pr-owl2:hasStateProbability  0f

```

Object Random Variable

Imagine we want to define the uncertainty that a person lives at some address. Suppose that there is already an OWL property that has that semantics (`Person livesAt Address`). Now, in order to define uncertainty about this property we need to create a random variable and map it to this OWL property. Since this property is a function (a person only lives at one address) we need to create a random variable (`RV.livesAt`), which defines the

uncertainty of the OWL property `livesAt`. Listing 4.4 presents this random variable. It has one argument, `RV.livesAt.MA.person`, its probability distribution is represented by `ex:RV.livesAt.DD.dist1`, and its range is `Address`.

Listing 4.4: Definition of the object random variable `livesAt(person)`

```

1 Individual: RV.livesAt
2   Types:
3     pr-owl2:RandomVariable
4   Facts:
5     pr-owl2:hasPossibleValues "&ex; Address"^^xsd:anyURI ,
6     pr-owl2:hasArgument  RV.livesAt.MA.person ,
7     pr-owl2:hasProbabilityDistribution  RV.livesAt.DD.dist1 ,
8     pr-owl2:definesUncertaintyOf "&ex;livesAt"^^xsd:anyURI

```

Until now we have just given names but no semantics to the argument and probability distribution of this random variable.

Listing 4.5: Argument of the object random variable `livesAt(person)`

```

1 Individual: RV.livesAt.MA.person
2   Types:
3     pr-owl2:MappingArgument
4   Facts:
5     pr-owl2:isArgumentOf  RV.livesAt ,
6     pr-owl2:hasArgumentNumber 1 ,
7     pr-owl2:isSubstitutedBy "&ex; Person"^^xsd:anyURI ,
8     pr-owl2:isSubjectIn "&ex;livesAt"^^xsd:anyURI

```

Listing 4.5 presents the semantics for the argument. Argument `RV.livesAt.MA.person` is the first and only argument of the random variable `RV.livesAt` and it can be substituted by the class `Person`. This argument is the subject in the OWL property `livesAt`.

Listing 4.6: Declarative distribution for the random variable `livesAt(person)`

```

1 Individual: RV.livesAt.DD.dist1
2   Types:
3     pr-owl2:DeclarativeDistribution
4   Facts:
5     pr-owl2:isProbabilityDistributionOf  RV.livesAt ,
6     pr-owl2:hasDeclaration "[Uniform();]"^^xsd:string ,
7     pr-owl2:isRepresentedAs "UnBBayes"^^xsd:string

```

Finally, Listing 4.6 presents a declarative distribution for the random variable `livesAt(person)` represented by the application-dependent format for local probability distribution defined in UnBBayes. The distribution is uniform over the possible values of the random variable (individuals of the class `Address`).

Data Random Variable

Imagine we want to define the uncertainty over how much a person makes per year. Suppose that there is already an OWL property that has that semantics (`Person hasAnnualIncome float`). For simplicity, let's assume the amount of income a person makes per year is represented as a float, instead of being another class which would have the float value and the unit like Dollar (for information on how to represent values and their units see discussion about quality and quantity in [93]). Now, in order to define uncertainty about this property we need to create a random variable and map it to this OWL property. Since this property is a function (a person only makes a certain amount of money per year) we need to create a random variable (`RV.hasAnnualIncome`), which defines the uncertainty of the OWL property `hasAnnualIncome`. Listing 4.7 presents this random variable. It has one argument, `RV.hasAnnualIncome.MA.person`, its probability distribution is represented by `ex:RV.hasAnnualIncome.DD.dist1`, and its range is `float`.

It is worth noting that in PR-OWL 1 such a random variable was not possible. This type of construction is one of the contributions of this work, due to the use of existing types supported by OWL.

Listing 4.7: Definition of the data random variable `hasAnnualIncome(person)`

```

1 Individual: RV.hasAnnualIncome
2   Types:
3     pr-owl2:RandomVariable
4   Facts:
5     pr-owl2:hasPossibleValues "&xsd; float"^^xsd:anyURI ,
6     pr-owl2:hasArgument RV.hasAnnualIncome.MA.person ,
7     pr-owl2:hasProbabilityDistribution RV.hasAnnualIncome.DD.dist1 ,
8     pr-owl2:definesUncertaintyOf "&ex; hasAnnualIncome"^^xsd:anyURI

```

Until now we have just given names but no semantics to the argument and probability distribution of this random variable.

Listing 4.8 presents the semantics for the argument. Argument `RV.hasAnnualIncome.MA.person` is the first and only argument of the random variable `RV.hasAnnualIncome` and it can be substituted by the class `Person`. This argument is the subject in the OWL property `hasAnnualIncome`.

Listing 4.8: Example of mapping argument

```

1 Individual: RV.hasAnnualIncome.MA.person
2   Types:
3     pr-owl2:MappingArgument
4   Facts:
5     pr-owl2:isArgumentOf RV.hasAnnualIncome ,
6     pr-owl2:isSubstitutedBy "&ex; Person"^^xsd:anyURI ,
7     pr-owl2:isSubjectIn "&ex; hasAnnualIncome"^^xsd:anyURI ,
8     pr-owl2:hasArgumentNumber 1

```

Finally, Listing A.13 presents a declarative distribution for the random variable `hasAnnualIncome(person)` represented by the application-dependent format for local probability distribution defined in UnBBayes. The distribution is normal with mean 50,000.00 and standard deviation 20,000.00.

Listing 4.9: Declarative distribution for the random variable `hasAnnualIncome(person)`

```

1 Individual: RV.hasAnnualIncome.DD.dist1
2   Types:
3     pr-owl2:DeclarativeDistribution
4   Facts:
5     pr-owl2:isProbabilityDistributionOf RV.hasAnnualIncome ,
6     pr-owl2:isRepresentedAs "UnBBayes"^^xsd:string ,
7     pr-owl2:hasDeclaration "[Normal(50000,20000);]"^^xsd:string

```

4.5 Entity Hierarchy and Polymorphism

One of the major changes in PR-OWL 2 was the removal of its own definition of entities. With this design decision, all entities used within PR-OWL 2 follow the same semantics defined in OWL. In other words, entities are defined as either classes or data types.

Moreover, since OWL supports multiple inheritance, so does PR-OWL 2. Thus, all the control over the type definition and type hierarchy in PR-OWL is delegated to OWL.

Nevertheless, there are situations where the PR-OWL reasoner will have to use this hierarchy information to decide which MFragment to instantiate. This is due to PR-OWL's 2 ability to overload probability distributions for the same random variable.

For instance, it is possible to define a distribution for the random variable `RV.hasHeight` given the domain is a `Person` and another distribution given the domain is a `VolleyballPlayer` (subclass of `Person`). So, at runtime, if someone is known only to be a person, then the first distribution will be used, however, if it is known that this person is in fact a volleyball player, then the second and more specific distribution will be used. In PR-OWL 1, since the input and context nodes were linked directly with a resident node, the user would be forced to choose one of the two distributions before runtime, during modeling. This was clearly not flexible enough to support polymorphism.

The polymorphism rules in PR-OWL 2 to decide which MFragment to instantiate in precedence order are:

1. Choose the MFragment where the resident node has the argument with the most specific class;
2. If more than one MFragment can be chosen from rule 1, then choose the MFragment with satisfied context nodes;
3. If more than one MFragment with satisfied context nodes exists, then the model is inconsistent;
4. If there is no MFragment with satisfied context nodes, but only one with unknown context nodes (not satisfied, but not unsatisfied either), then choose this MFragment;

5. Otherwise, use the default distribution for that random variable.

To clarify the rules presented, let's extend the example given previously. Imagine we have the following hierarchy structure⁵:

```
• Animal
  - IntelligentAnimal
    * Person
      · VolleyballPlayer
  - NonIntelligentAnimal
    * Biped
      · Bird
    * Quadruped
      · Horse
      · Dog
```

Imagine now that besides having a distribution for the height of a person and a volleyball player, we also have MFrags with distributions for the height of bipeds, quadrupeds, birds, and horses, but not of dogs.

As explained before, if the individual is a person, then we should use the distribution for person. However, if we also know that this individual is a volleyball player, then because of rule 1, the more specific distribution should be used, which is the distribution for volleyball players.

In cases like the one just described, where we define a probability for a more specific class (like height for volleyball players) and also for a less specific class (like height for persons) then the correct thing to do is to interpret the probability for the less specific class as applying only to individuals that do not satisfy the more specific constraints.

Therefore, the correct rule would be:

⁵This is not supposed to be a good example of ontology engineering. The focus here is to present a really simple example just to illustrate the use and behavior of polymorphism.

- If it is known that Bill is a volleyball player, use $Normal(1.9, 0.2)$;
- If it is known that Bill is not a volleyball player, use $Normal(1.7, 0.2)$;
- If it is only known that Bill is a person, then use $Pr(VolleyballPlayer) * Normal(1.9, 0.2) + (1 - Pr(VolleyballPlayer)) * Normal(1.7, 0.2)$.

This requires us to specify a subclass probability if we want to define probabilities for a more specific subclass and are not always going to know for sure whether or not an individual belongs to the subclass. A reasoner would give an error message that the probability is undefined if we said Bill was a person and had not defined a probability that a person was a volleyball player.

Now, we could have an approximate reasoning method that used the Person distribution when we don't know that the person isn't a volleyball player. That is, we might allow a reasoner to adopt a "negation by failure" rule as an approximation. This would be an accurate approximation if the subclass had a low enough base rate and we didn't have strong evidence for belonging to the subclass. This approximation just described is the approach defined in the polymorphism rules presented above.

To better illustrate why this is an approximation, imagine the following example:

Case 1 : Suppose we know Joe is a volleyball player. Then we choose the MFragment with the most specific class, and his height is $Normal(1.9, 0.2)$ meters.

Case 2 : Suppose we know Fred is not a volleyball player. The most specific class is Person, which in our example is $Normal(1.7, 0.2)$ meters.

Case 3 : Suppose we know Bill is a person. Then according to the polymorphism rules described above his height would be $Normal(1.7, 0.2)$ meters.

The problem is that this would imply there is a zero percent chance that Bill is a volleyball player. This is because the rules of probability theory imply that his mean height is $1.9 * Pr[isA(Bill, VolleyballPlayer)] + 1.7 * (1 - Pr[isA(Bill, VolleyballPlayer)])$

The only way to satisfy this equation is if $Pr[isA(Bill, VolleyballPlayer)] = 0$.

Therefore, unless we know Bill is a volleyball player, or have directly assigned a non-zero probability to his being a volleyball player, then we are effectively assuming he cannot be a volleyball player.

The same logic applies for the non-intelligent animals. So, using the approximation rules defined above, if we know an individual is a biped, we should use the distribution for bipeds, however, if we know that this individual is also a bird, then because of rule 1, the more specific distribution should be used, which is the distribution for birds.

Nevertheless, if we know that the individual is a dog, the most specific distribution (rule 1) is the height of quadrupeds, since every dog is quadruped and there is no distribution for the height of dogs.

Now, imagine that there are two different distributions for the height of persons. In one there is a context node that says that the person has to be from Holland (known to be tall people) and the other says that the person has to be from Japan (known to be less tall). Looking just at the arguments of the resident nodes is not enough to decide which Mfrag to use, i.e., rule 1 cannot be used. However, if we now have an individual we know is from Holland, because of rule 2, we should use the first Mfrag (since it is the only one with satisfied context nodes). On the other hand, if we know that the individual is from Japan, because of rule 2, we should use the second Mfrag (since it is the only one with satisfied context nodes).

However, if for some reason this person has dual nationality being both from Holland and from Japan, because of rule 3, the model is now considered inconsistent. In other words, there is no way, based on the model, of deciding which Mfrag is the correct one. If this was allowed and an Mfrag was chosen, for instance, at random, then it would be possible to have different results based on the same model and knowledge base, which is inconsistent. Nevertheless, an implementation of PR-OWL 2 could allow the user to define rules on how to solve this kind of conflict. However, this is optional and is not covered by PR-OWL 2 semantics.

Suppose that we also have a random variable with the distribution for weight of animals.

However, suppose there is only one resident node with a specific distribution for weight of persons. If we have an individual we only know is an animal, although there is no MFragment with weight resident node that has satisfied context nodes, there is only one MFragment that has the weight resident node with unknown context nodes. Therefore, because of rule 4, this MFragment can be chosen by instantiating the type context node as an uncertain variable and parent of the nodes in that MFragment, stating that if this animal is in fact a person then use the distribution defined in this MFragment, otherwise, use the default distribution for the random variable weight.

Assume now that the individual is in fact a house and we know that a house cannot be an animal (disjoint from). Suppose also that there is no other distribution for heights than the ones already presented. Then, clearly, there is no MFragment with height resident node that has satisfied context nodes (they have, in fact, unsatisfied context nodes). Therefore, because of rule 5, the default distribution for the height random variable should be used.

Finally, if we only know that an individual is an animal, but we have no further information on what kind of animal it is, it is impossible to choose from the different distributions available. Should we instantiate the MFragment that has the distribution on heights of persons or should we instantiate the MFragment that has the distribution on heights of horses? Note that, different than the previous example, the MFragments do not have unsatisfied context nodes, in fact, they are uncertain. However, since more than one MFragment could be perfectly valid by instantiating the type context node as an uncertain variable and parent of the nodes in that MFragment, there is no correct answer. Therefore, the default behavior should be to use the default distribution as defined in rule 5.

4.6 Type Uncertainty

A special kind of built-in Boolean random variable is `isA(resource, class)`, which represents type, whereas its probability distribution represents type uncertainty. As expected, this RV defines the uncertainty of the RDF property `rdf:type`. Listing 4.10 presents its

definition. The domain is defined by two arguments, the object (`rdf:Resource`), represented by the first argument `isA.resource`, and the type (`rdf:Class`), represented by the second argument `isA.class`. Its range is Boolean (inferred from its type `BooleanRandomVariable`). Finally, its probability distribution is left in blank and the PR-OWL 2 reasoner should allow the knowledge engineer to define this distribution. The objective is to give flexibility to the user in deciding the best default distribution to use (*e.g.*, uniform distribution, or a fixed low probability for `true` and the rest for `false`).

Listing 4.10: Built-in Boolean random variable for type uncertainty (`isA(resource, class)`)

```

1 Individual: isA
2   Types:
3     BooleanRandomVariable
4   Facts:
5     hasArgument  isA.class ,
6     hasArgument  isA.resource ,
7     definesUncertaintyOf  "&rdf:type"^^xsd:anyURI
8
9 Individual: isA.resource
10  Types:
11    MappingArgument
12  Facts:
13    isArgumentOf  isA ,
14    isSubstitutedBy  "&rdf:Resource"^^xsd:anyURI ,
15    isSubjectIn  "&rdf:type"^^xsd:anyURI ,
16    hasArgumentNumber  1
17
18 Individual: isA.class
19  Types:
20    MappingArgument
21  Facts:
22    isArgumentOf  isA ,
23    isObjectIn  "&rdf:type"^^xsd:anyURI ,
24    isSubstitutedBy  "&rdf:Class"^^xsd:anyURI ,
25    hasArgumentNumber  2

```

When defining the types of ordinary variables or exemplars to use in MFrags, the property used should be the `isSubstitutedBy`. However, when evaluating the context node, if it is not known whether a given individual is or is not of that specific type defined by the property `isSubstitutedBy`, then the reasoner must look for an Mfrag with the resident node `isA` for that unknown type. For instance, suppose that we have a random variable with the

distribution for weight of animals. However, suppose there is only one resident node with a specific distribution for weight of persons (*i.e.*, there is an ordinary variable `person`, where `person isSubstitutedBy Person`). Assume also that `Person rdfs:subClassOf Animal`. If we have an individual we only know is an animal, although there is no MFragment with weight resident node that has satisfied context nodes, there is only one MFragment that has the weight resident node with unknown context nodes (the `isA(person, Person)`, represented by the ordinary variable `person`, where `person isSubstitutedBy Person`). Therefore, this MFragment can be chosen by instantiating the type context node (`isA(person, Person)`) as an uncertain variable and parent of the nodes in that MFragment, stating that if this animal is in fact a person then use the distribution defined in this MFragment, otherwise, use the default distribution for the random variable weight. (see Section 4.5 for more information on entity hierarchy and polymorphism).

Although there is no built-in random variable for subtypes in PR-OWL 2, the PR-OWL reasoner must ensure that if it is known that class A is subtype of class B, then an individual `a` of class A is also of type B, *i.e.*, `isA(a,B)=true`.

4.7 Defining Nodes in PR-OWL 2

Figure 4.18 presents the main MEBN elements and their relations. A MEBN theory is composed of one or more MEBN fragments. A MEBN fragment is composed of one or more nodes and it can also have ordinary variables and exemplars, which are used in first-order logic formulas (represented by MEBN expressions). A node has its representation defined by a MEBN expression. An MEBN expression is a first-order logic formula or term, of a specific type, represented by the random variable (*e.g.*, an equal formula has the `equalTo` random variable as its type). A random variable has a default probability distribution. Finally, there are three types of nodes. A resident node can be parent of another resident node and has a probability distributions associated with it. An input node represents a first-order formula or term that influences the distribution of at least one resident node, but

has its distribution defined by the random variables it uses (not defined within that MEBN fragment). Finally, context nodes are first-order formulas that have to be true in order for the probability distributions defined within their MEBN fragment to be valid.

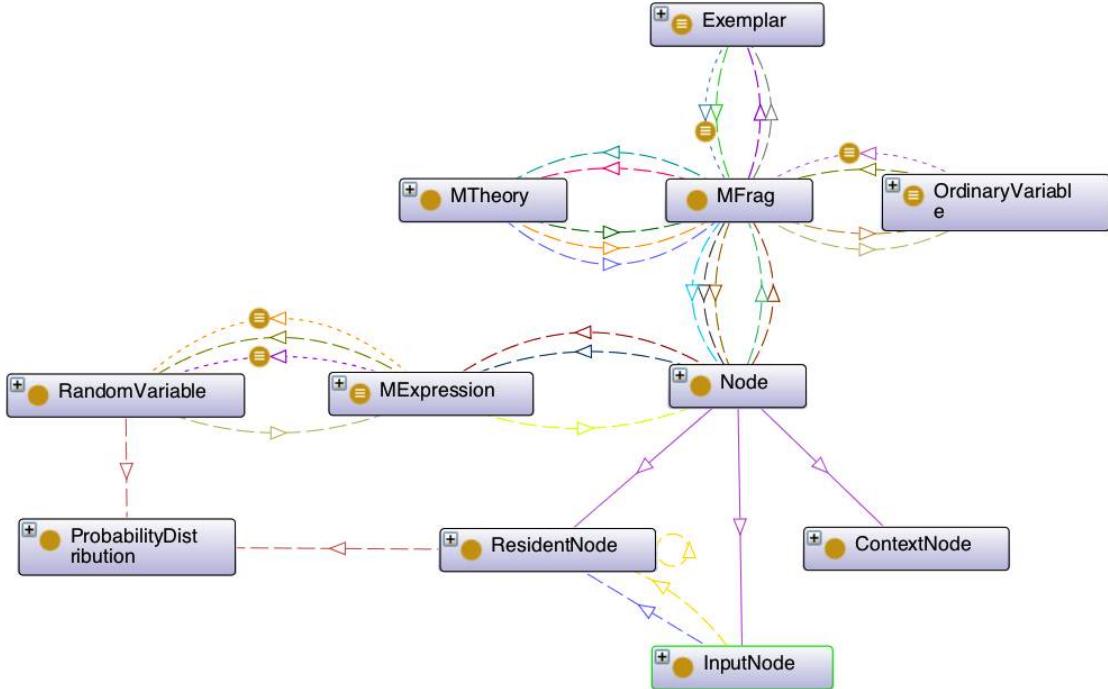


Figure 4.18: Graph of the main MEBN elements and their relations.

A **Node** is part of an **MFragment** and it can define the distribution of a random variable within that **MFragment** (a resident node, represented by the class **ResidentNode**), a random variable that influences the distribution of nodes within that **MFragment** but has its distribution defined somewhere else (an input node, represented by the class **InputNode**), or a random variable that expresses the context in which the probability distributions within that **MFragment** are valid (a context node, represented by the class **ContextNode**). **ResidentNode**, **InputNode**, and **ContextNode** are disjoint classes.

In all cases, the random variable represented by the node is defined as a MEBN expression (see Subsection 4.7.3 for more information). Besides that, every node is resident in

exactly one MFragment.

Since there were not major changes in the definition of MTheories, MFrags, and nodes in PR-OWL 2, except the use of MEBN expressions, we will briefly describe them in Sub-sections 4.7.1 and 4.7.2. For detailed information about these concepts see Appendix A Section A.2. Then, Subsection 4.7.3 will present the main concepts on defining MEBN expressions, however, for further detail we refer the reader to Appendix A Section A.3. Finally, Subsection 4.7.4 present different examples of domain-specific nodes and finding nodes.

4.7.1 Defining Domain-Specific Knowledge

All generative MFrags created by the ontology engineer (*i.e.* the domain expert) are individuals of `DomainMFragment`. A `DomainMFragment` is the subclass of class `MFragment` that includes all the domain-specific MFrags. It is disjoint from the class `FindingMFragment`.

Figure 4.19 presents the main concepts in defining a domain-specific MFragment and its relations. A domain-specific MFragment, besides exemplars and ordinary variables, has more specific input nodes and resident nodes (namely, `GenerativeInputNode` and `DomainResidentNode`, respectively). A domain resident node only accepts simple MEBN expressions, while generative input nodes accept any type of MEBN expression. Besides input nodes and resident nodes, a domain MFragment has context nodes, which are responsible for defining the restrictions that must be satisfied in order for the probabilistic relations and distributions within that MFragment to be valid. As a context node represents a constraint that must be true, it only accepts Boolean MEBN expressions.

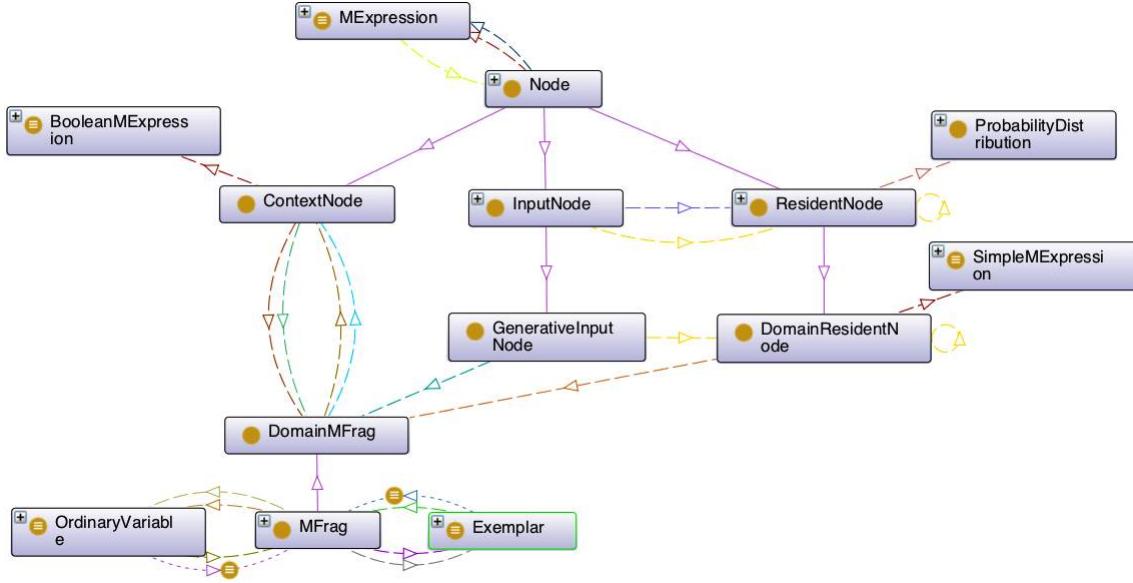


Figure 4.19: Graph with the main elements necessary to define a domain-specific MEBN fragment.

A `DomainResidentNode` is the subclass of `ResidentNode` that includes all domain-specific resident nodes. It is disjoint from `FindingResidentNode`. It only defines a distribution over a simple MEBN expression. It can only have parents of type `GenerativeInputNode` or `DomainResidentNode`. Finally, a `DomainResidentNode` can only be defined in exactly one `DomainMfrag`.

A `GenerativeInputNode` is a random variable that has its distribution defined somewhere else, but its value influences some `DomainResidentNode` within that `Mfrag`. It can only be an input node in exactly one `DomainMfrag`. Finally, it is disjoint from `FindingInputNode`.

A `ContextNode` defines a constraint on the `Mfrag` where it is defined. Often these constraints define the type of arguments used for resident nodes (*e.g.*, `isA(mother, Person)`, `isA(person, Person)`), reference constraints between these arguments (*e.g.*, `mother = motherOf(person)`), etc. It is disjoint from `ResidentNode` and `InputNode`. It can only represent a Boolean MEBN expression (see `BooleanMExpression` in Subsection 4.7.3 for

more information), since it is being stated that this Boolean expression, the constraint, has to be true (satisfied). Finally, it can only be a context node in exactly one **DomainMfrag**.

4.7.2 Defining Findings

A **FindingMfrag** is used to convey information about findings, which is the default way of entering evidence in an MEBN theory so a probabilistic algorithm can be applied to perform inferences regarding the new evidence. It has no context nodes, only one input and one resident node.

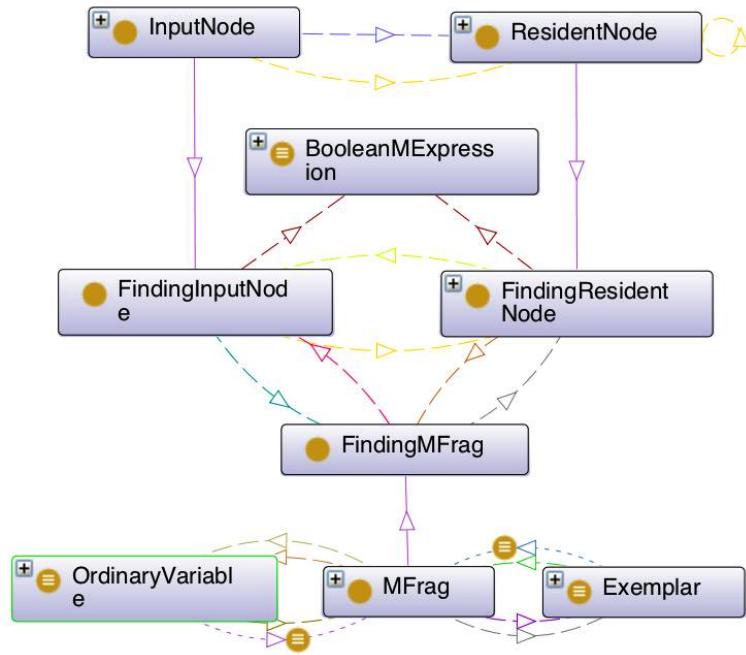


Figure 4.20: Graph with the main concepts and their relations for defining findings in an MEBN theory.

Figure 4.20 presents the main concepts involved in entering findings in an MEBN theory. Besides having ordinary variables and exemplars that can be used in formulas, which defines the finding, a finding Mfrag only has one finding resident node and one finding input node. Since the idea of a finding is to say that some formula is true, the only MEBN expression

allowed for finding resident nodes and input nodes is Boolean.

The MEBN representation of a finding is shown in Figure 4.21. It can be seen that both the finding resident and input nodes represent the same Boolean expression. The difference is that a finding resident node only has true and absurd as possible values and its local probability distribution is defined as true if the input node is true and absurd otherwise. The finding input node has the same behavior as any regular input node, except that it can only represent Boolean expressions.

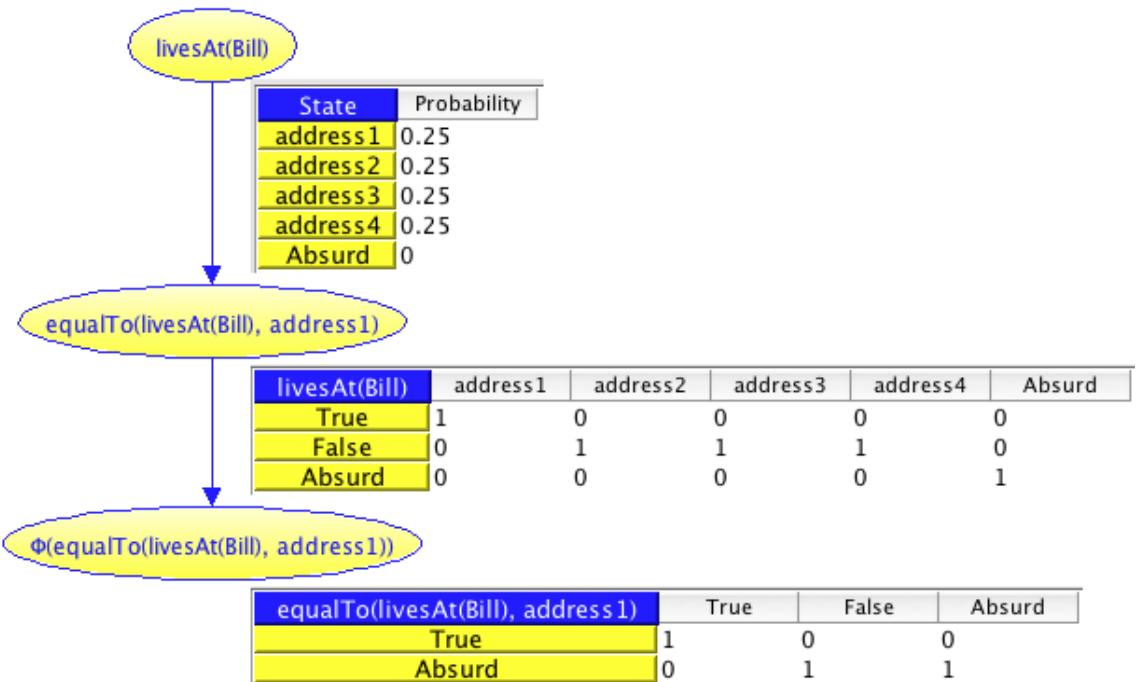


Figure 4.21: Bayesian network showing the pattern for representing findings in MEBN.

A `FindingResidentNode` is the subclass of `ResidentNode` that includes all finding nodes. Finding nodes convey new evidence into a probabilistic system via a `FindingM-Frag`. It is disjoint from `DomainResidentNode`. It can only represent a Boolean MEBN expressions (see `BooleanMExpression` in Subsection 4.7.3 for more information), since it is being stated that this Boolean expression is true (a finding). It has only one parent and it

has to be of the type `FindingInputNode`. It cannot be parent of any other node. Finally, it can only be defined in exactly one `FindingMfrag`.

A `FindingInputNode` represents a Boolean MEBN expression (see `BooleanMExpression` in Subsection 4.7.3 for more information), which influences some `FindingResidentNode` within that `Mfrag`. In fact, it can only be parent of one node, and this node has to be of type `FindingResidentNode`. It can only be an input node in exactly one `FindingMfrag`. Finally, it is disjoint from `GenerativeInputNode`.

4.7.3 MEBN Expressions

An `MExpression` represents a first-order logic formula or term, which has the random variable as its main element (*e.g.*, `equalTo`, `or`, `and`, `livesAt`, etc). Figure 4.22 presents the main concepts and their relations necessary for defining MEBN expressions and their arguments. The number of arguments defined on the MEBN expression has to be the same as the number of arguments defined on the RV it refers to. Furthermore, the argument types have to be compatible. Every node is represented by an MEBN expression. However, some nodes can only be represented by a specific type of `MExpression`. `FindingResidentNode`, `FindingInputNode`, and `ContextNode` can only be represented by `BooleanMExpression`. A `BooleanMExpression` only allow a specific type of random variable, `BooleanRandomVariable`. `DomainResidentNode` can only be represented by `SimpleMExpression`, which is an MEBN expression that only has arguments of type `OrdinaryVariable` or `ConstantArgument`, since it represents an atomic formula or term.

There are mainly two different types of arguments:

1. those used for mapping random variable arguments to OWL properties domain or range (represented by the class `MappingArgument`, which is discussed in Subsection 4.4.4), and
2. those used in MEBN expressions (`MExpressionArguemnt`, `ExemplarArgument`, `OrdinaryVariableArgument`, and `ConstantArgument`).

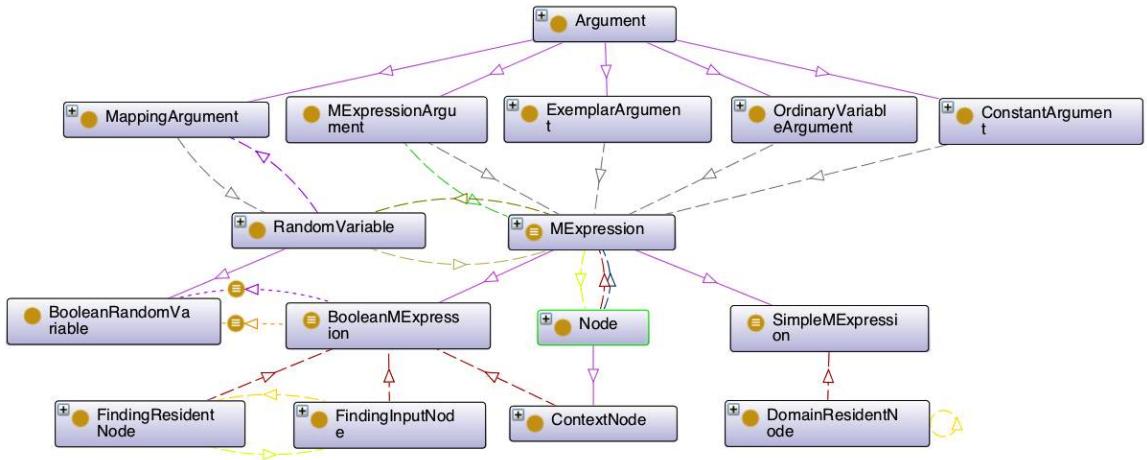


Figure 4.22: Graph with main concepts and their relations necessary for defining arguments and MEBN expressions.

A **ConstantArgument** is used to represent formulas or terms which use either data and/or object constants, *e.g.*, `equalTo(livesAt(Bill), address1)` (where Bill and address1 are constants, which represent Person and Address, respectively), `equalTo(hasAnnualIncome(Bill), 75,000.00)` (assuming income is just a number, which represents value in US Dollar and Bill and 75,000.00 are constants, which represent Person and float, respectively).

An **OrdinaryVariableArgument** is used to represent free variable arguments (not quantified over) used in a formula or term, *e.g.*, `livesAt(person)`, where person is a free variable, that can be substituted by an individual of the class Person.

An **ExemplarArgument** is used to represent a filler for a bound variable (variables that are quantified over) used in a formula, *e.g.*, `forAll(mother) (forAll(child) (implies(hasChild(mother,child), isRelated(child,mother))))`, where mother and child are bound variables of type Person.

Finally, an **MExpressionArgument** is used to allow the construction of complex formulas or terms (more than one RV used in the formula or term), *e.g.*,

`equalTo(livesAt(person1), livesAt(person2))`, where `person1` and `person2` are free variables of type `Person`.

4.7.4 Examples of Nodes

In the following subsections we will present different examples of nodes. First, we present examples of nodes to allow the specification of domain-specific knowledge for defining that two people are more likely to be related if they live at the same address. Then, we present examples of nodes to allow the specification of the evidence that two people live at the same address.

Domain-Specific Nodes

Figure 4.23 presents a domain-specific MFragment for representing the probabilistic knowledge that two people are more likely to be related if they live at the same address.

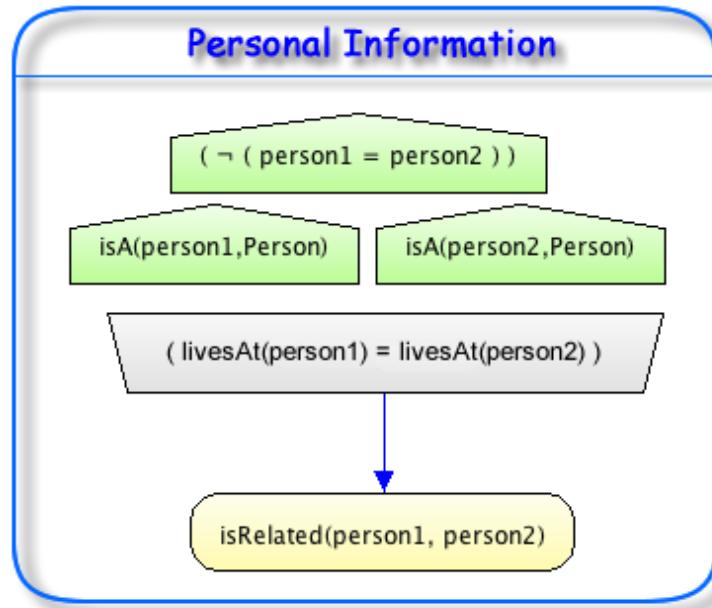


Figure 4.23: Nodes for representing that two people are more likely to be related if they live at the same address.

Before we get into the details of how to represent such MFragment in PR-OWL 2, it is important to discuss property constraints or restrictions such as symmetry, since `isRelated`, which is discussed in this MFragment, is a symmetric property.

In OWL there are various restrictions, like symmetry, that one can assign to properties. In order to represent such restrictions in PR-OWL, we first need to understand how to represent them in regular BNs.

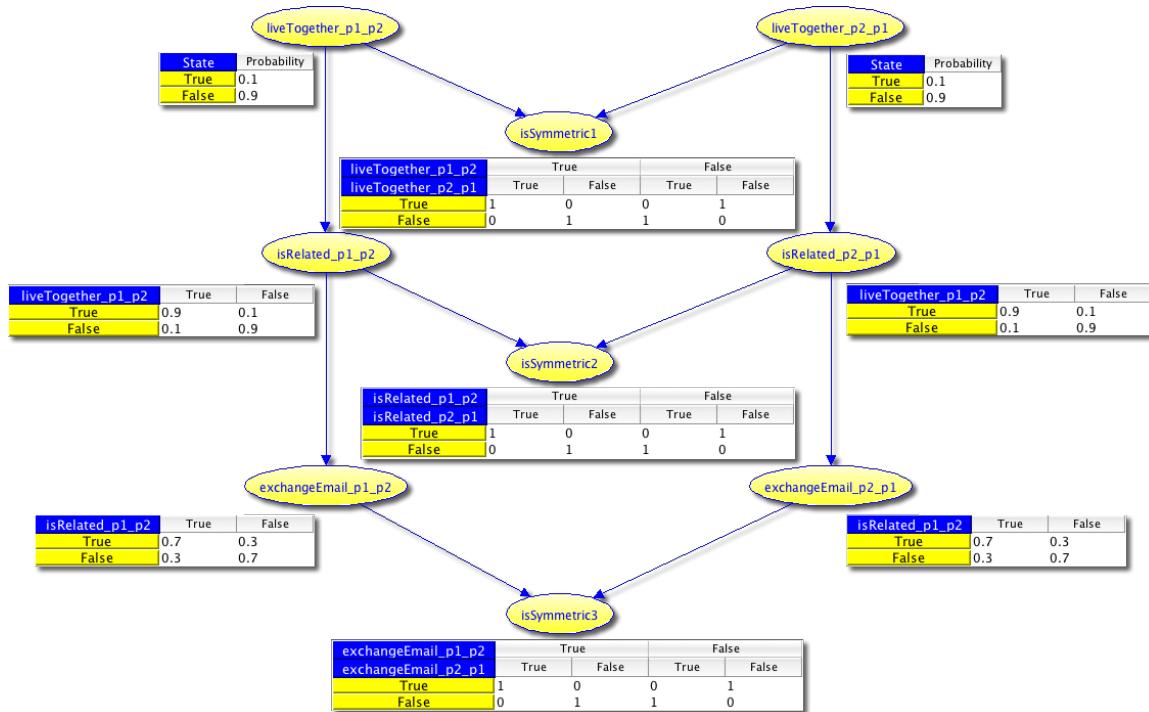
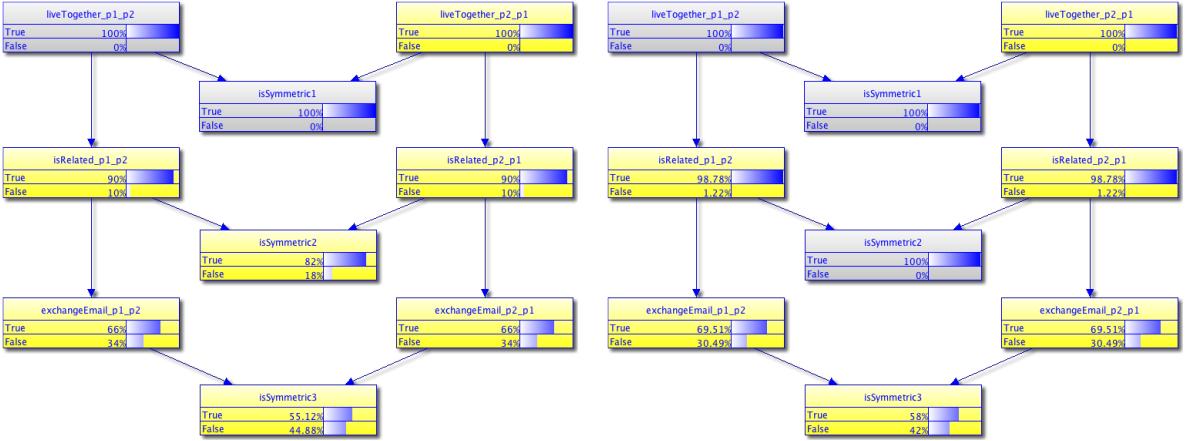


Figure 4.24: A common way to define restrictions like symmetry in BNs.

Figure 4.24 presents a common way to represent constraints like symmetry in BNs. The relations `liveTogether`, `isRelated`, and `exchangeEmail` are symmetrical. These symmetries are represented by stating that the nodes `isSymmetric1`, `isSymmetric2`, and `isSymmetric3` are true, respectively. The symmetrical restriction is defined by the CPT, where the restriction is true if both parents are true or if both are false.



(a) Posterior probability of symmetrical property `liveTogether` using symmetry constraint node.
(b) Double counting of symmetrical evidence.

Figure 4.25: Posterior probabilities for symmetrical properties showing how the constraint works, but unfortunately it double counts evidence.

Figure 4.25(a) shows that this actually works. By saying that `liveTogether` is a symmetrical property through the evidence in `isSymmetric1`, we can see that if we say that `liveTogether_p1_p2` is true, the node `liveTogether_p2_p1` automatically gets the same value, *i.e.*, 100% for the state true. However, as we can see in Figure 4.25(b) if we say that `isRelated` is also a symmetrical property through the evidence in `isSymmetric2`, we can see that the posterior for both `isRealted_p1_p2` and `isRelated_p2_p1` change, which should not be the case. This happens because the evidence that p1 and p2 live together is double counted. So it is clear that this solution cannot be applied, otherwise we would double count every time we have a restriction like this one in our model.

Figure 4.26 presents another solution for the same problem. Now, instead of having a node defining the symmetrical restriction, an order is defined amongst the arguments of the relation. For instance, we assume that there is an order in instances of `Person` and that `p1` comes before `p2`. Once we have the order, we use this order to define which node gets the normal distribution defined by the CPT of the property and which node gets the CPT of the restriction itself. In our scenario, we can see that all nodes that have arguments `p1_p2`

get the regular CPT defined for their relation and the nodes that have arguments p2_p1 get the CPT of the restriction, *i.e.*, true if the parent is true and false if the parent is false. This will guarantee that they will have the same value, independent of where the evidence is entered.

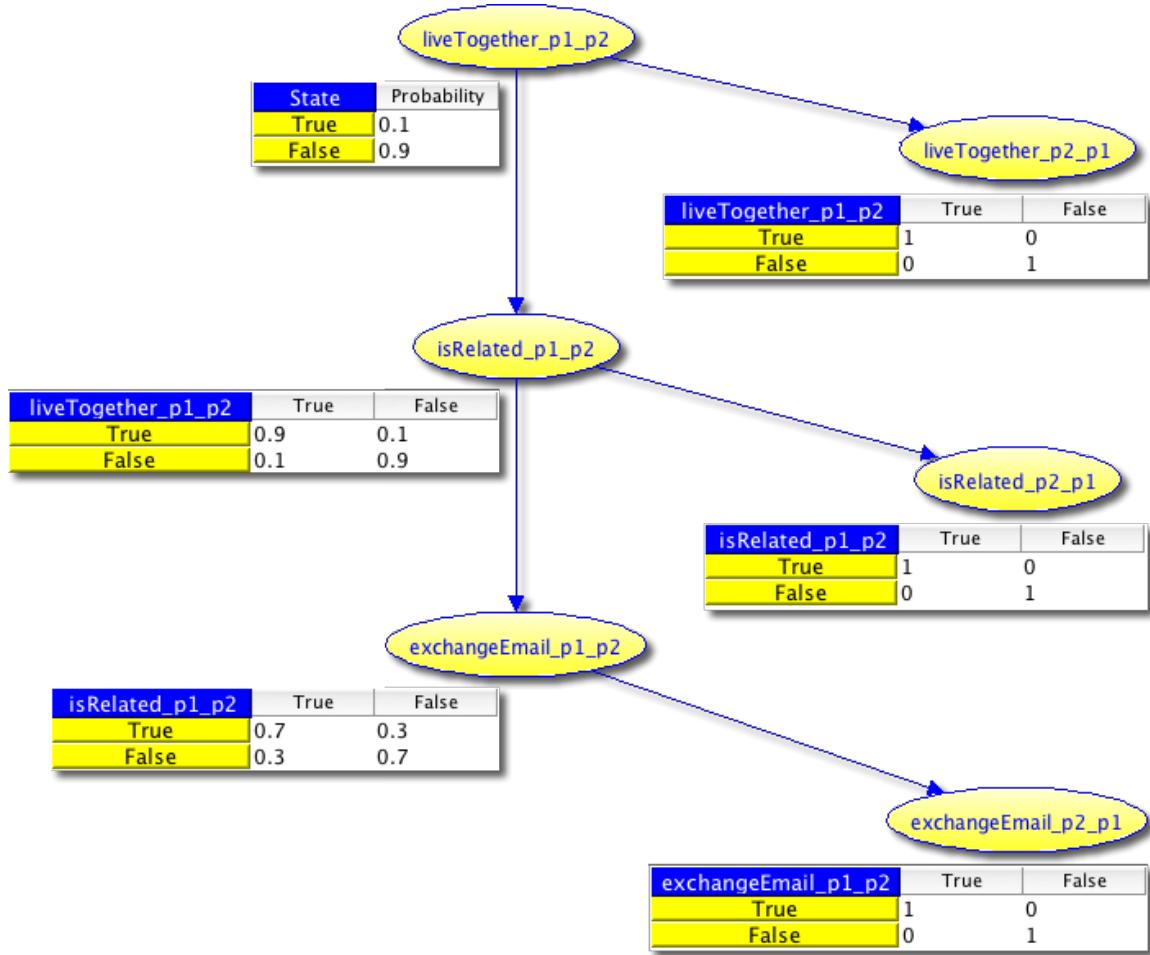


Figure 4.26: Defining restrictions like symmetry in BNs using order on arguments.

Figure 4.27 shows that this actually works. By saying that that `liveTogether_p1_p2` is true, the node `liveTogether_p2_p1` automatically gets the same value, *i.e.*, 100% for the state true. Furthermore, the relation `isRelated` is already mapped as a symmetrical property, however, this does not affect the posterior as it happened in the previous solution.

We can see that the posterior for both `isRealted_p1_p2` and `isRelated_p2_p1` are the same and still 90%, as expected.

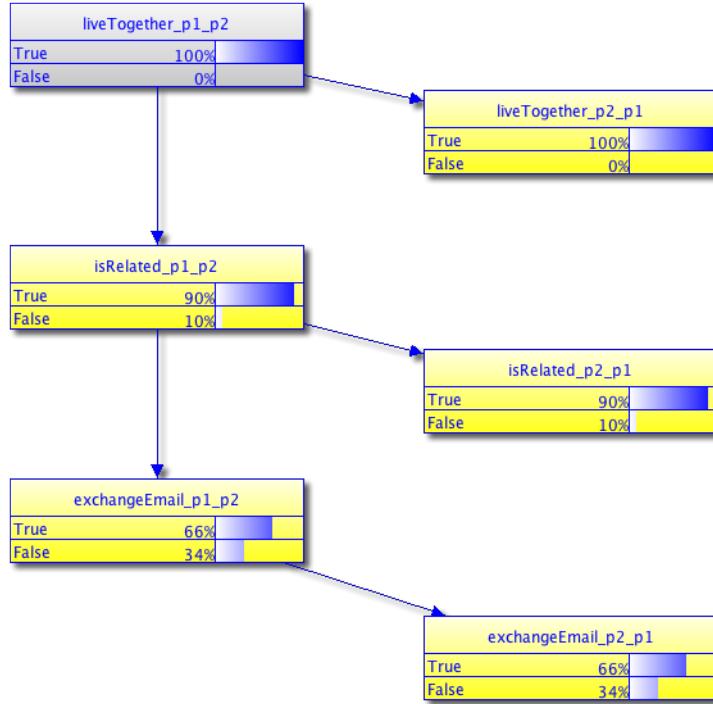


Figure 4.27: Posterior probabilities for symmetrical properties using order on arguments, which does not cause double counting of evidence.

The solution just presented can be generalized in order to apply them to PR-OWL random variables. However, this will not be included as a built-in feature in PR-OWL 2. Nevertheless, PR-OWL 2 is expressive enough to allow the user to define such restrictions, the same way existential uncertainty is not built-in, but can be easily represented in the language (see Section 4.8 for further details on existential uncertainty).

Listing 4.11 presents an Mfrag called `PersonalInformation`, that defines the distribution of two different people being related given they live at the same address, as seen in Figure 4.23.

Listing 4.11: Domain MFragment for representing that two people are more likely to be related if they live at the same address

```

1 Individual: MFragment. PersonalInformation
2   Types:
3     pr-owl2:DomainMFragment
4   Facts:
5     pr-owl2:isMFragmentOf MTheory. FraudIdentificationInPublicProcurement ,
6     pr-owl2:hasOrdinaryVariable MFragment. PersonalInformation .OV. person1 ,
7     pr-owl2:hasOrdinaryVariable MFragment. PersonalInformation .OV. person2 ,
8     pr-owl2:hasContextNode MFragment. PersonalInformation .CN. not1 ,
9     pr-owl2:hasInputNode MFragment. PersonalInformation .GIN. equalTo1 ,
10    pr-owl2:hasResidentNode MFragment. PersonalInformation .DRN. isRelated

```

Listing 4.12 presents the ordinary variables `person1` and `person2`, which can be substituted by individuals of the class `Person`. These ordinary variables are defined in the MFragment `PersonalInformation`. Finally, these ordinary variables are used as arguments in three different MEBN expressions, which represent nodes in their MFragment.

Listing 4.12: Ordinary variables used to represent two different persons in the same domain MFragment

```

1 Individual: MFragment. PersonalInformation .OV. person1
2   Types:
3     pr-owl2:OrdinaryVariable
4   Facts:
5     pr-owl2:isOrdinaryVariableIn MFragment. PersonalInformation ,
6     pr-owl2:isTypeOfArgumentIn MFragment. PersonalInformation .CN. not1 .BME. not1 .BME
7       .equalTo1 .OVA. person1 ,
8     pr-owl2:isTypeOfArgumentIn MFragment. PersonalInformation .GIN. equalTo1 .BME.
9       .equalTo1 .ME. livesAt1 .OVA. person1 ,
10    pr-owl2:isTypeOfArgumentIn MFragment. PersonalInformation .DRN. isRelated .SME.
11      .isRelated .OVA. person1 ,
12      pr-owl2:isSubstitutedBy "&ex; Person" ^^xsd:anyURI
13
14 Individual: MFragment. PersonalInformation .OV. person2
15   Types:
16     pr-owl2:OrdinaryVariable
17   Facts:
18     pr-owl2:isOrdinaryVariableIn MFragment. PersonalInformation ,
19     pr-owl2:isTypeOfArgumentIn MFragment. PersonalInformation .CN. not1 .BME. not1 .BME
       .equalTo1 .OVA. person2 ,
20     pr-owl2:isTypeOfArgumentIn MFragment. PersonalInformation .GIN. equalTo1 .BME.
       .equalTo1 .ME. livesAt1 .OVA. person2 ,
21     pr-owl2:isTypeOfArgumentIn MFragment. PersonalInformation .DRN. isRelated .SME.
       .isRelated .OVA. person2 ,
22     pr-owl2:isSubstitutedBy "&ex; Person" ^^xsd:anyURI

```

Listing 4.13 presents a context node for the Boolean expression `not(equalTo(person1, person2))`. In other words, this context node states that within its MFragment, `person1` and `person2` have to be different.

Listing 4.13: Context node example

```

1 Individual: MFragment.PersonalInformation.CN.not1
2   Types:
3     pr-owl2:ContextNode
4   Facts:
5     pr-owl2:isContextNodeIn MFragment.PersonalInformation,
6     pr-owl2:hasMExpression MFragment.PersonalInformation.CN.not1.BME.not1

```

Listing 4.14 presents how the Boolean expression `not(equalTo(person1, person2))` for the context node `MFragment.PersonalInformation.CN.not1` is defined.

Listing 4.14: Boolean expression for the context node `not(equalTo(person1, person2))`

```

1 <!-- The Boolean expression not(equalTo(person1, person2)) -->
2 Individual: MFragment.PersonalInformation.CN.not1.BME.not1
3   Types:
4     pr-owl2:BooleanMExpression
5   Facts:
6     pr-owl2:typeOfMExpression pr-owl2:not ,
7     pr-owl2:hasArgument MFragment.PersonalInformation.CN.not1.BME.not1.MEA.arg1 ,
8     pr-owl2:isMExpressionOf MFragment.PersonalInformation.CN.not1
9
10 <!-- The first and only argument equalTo(person1, person2) of not(equalTo(
11   person1, person2)) -->
12 Individual: MFragment.PersonalInformation.CN.not1.BME.not1.MEA.arg1
13   Types:
14     pr-owl2:MExpressionArgument
15   Facts:
16     pr-owl2:isArgumentOf MFragment.PersonalInformation.CN.not1.BME.not1 ,
17     pr-owl2:typeOfArgument MFragment.PersonalInformation.CN.not1.BME.not1.BME.
18       equalTo1 ,
19     pr-owl2:hasArgumentNumber 1
20 <!-- The type of the first argument, which is the expression equalTo(person1,
21   person2) -->
22 Individual: MFragment.PersonalInformation.CN.not1.BME.not1.BME.equalTo1
23   Types:
24     pr-owl2:BooleanMExpression
25   Facts:
26     pr-owl2:typeOfMExpression pr-owl2:equalTo ,
27     pr-owl2:hasArgument MFragment.PersonalInformation.CN.not1.BME.not1.BME.
28       equalTo1.OVA.person1 ,
29     pr-owl2:hasArgument MFragment.PersonalInformation.CN.not1.BME.not1.BME.
30       equalTo1.OVA.person2 ,

```

```

27 pr-owl2:isTypeOfArgumentIn Mfrag. PersonalInformation .CN. not1 .BME. not1 .MEA
28     .arg1
29 <!-- The first argument person1 of the expression equalTo(person1, person2) --
30     -->
31 Individual: Mfrag. PersonalInformation .CN. not1 .BME. not1 .BME. equalTo1 .OVA.
32     person1
33 Types:
34     pr-owl2:OrdinaryVariableArgument
35 Facts:
36     pr-owl2:typeOfArgument Mfrag. PersonalInformation .OV. person1 ,
37     pr-owl2:isArgumentOf Mfrag. PersonalInformation .CN. not1 .BME. not1 .BME.
38         equalTo1 ,
39     pr-owl2:hasArgumentNumber 1
40
41 <!-- The second argument person2 of the expression equalTo(person1, person2) --
42     -->
43 Individual: Mfrag. PersonalInformation .CN. not1 .BME. not1 .BME. equalTo1 .OVA.
44     person2
45 Types:
46     pr-owl2:OrdinaryVariableArgument
47 Facts:
48     pr-owl2:typeOfArgument Mfrag. PersonalInformation .OV. person2 ,
49     pr-owl2:isArgumentOf Mfrag. PersonalInformation .CN. not1 .BME. not1 .BME.
50         equalTo1 ,
51     pr-owl2:hasArgumentNumber 2

```

Listing 4.15 presents the generative input node for the MEBN expression `equalTo(livesAt(person1), livesAt(person2))` defined for the Mfrag `Mfrag.PersonalInformation`. It is parent of the resident node `isRelated(person1, person2)`. This node state that `person1` and `person2` live at the same address and is used to influence the distribution of these two people being related.

Listing 4.15: Generative input node `equalTo(livesAt(person1), livesAt(person2))`

```

1 Individual: Mfrag. PersonalInformation .GIN. equalTo1
2     Types:
3         pr-owl2:GenerativeInputNode
4     Facts:
5         pr-owl2:isInputNodeIn Mfrag. PersonalInformation ,
6         pr-owl2:isParentOf Mfrag. PersonalInformation .DRN. isRelated
7         pr-owl2:hasMExpression Mfrag. PersonalInformation .GIN. equalTo1 .BME.
              equalTo1

```

Listing 4.16 presents how the Boolean expression

`equalTo(livesAt(person1), livesAt(person2))` for the generative input node `MFrag.PersonalInformation.GIN.equalTo1` is defined.

Listing 4.16: Boolean expression for the generative input node `equalTo(livesAt(person1), livesAt(person2))`

```

1  <!-- The Boolean expression equalTo(livesAt(person1), livesAt(person2)) -->
2  Individual: MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1
3  Types:
4      pr-owl2:BooleanMExpression
5  Facts:
6      pr-owl2:typeOfMExpression pr-owl2:equalTo ,
7          pr-owl2:hasArgument MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1 .
8              MEA.arg2 ,
9          pr-owl2:isMExpressionOf MFrag.PersonalInformation.GIN.equalTo1 ,
10         pr-owl2:hasArgument MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1 .
11             MEA.arg1
12
13 <!-- The first argument livesAt(person1) of equalTo(livesAt(person1), livesAt(
14     person2)) -->
15 Individual: MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1.MEA.arg1
16 Types:
17     pr-owl2:MExpressionArgument
18 Facts:
19     pr-owl2:isArgumentOf MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1 ,
20     pr-owl2:typeOfArgument MFrag.PersonalInformation.GIN.equalTo1.BME.
21         equalTo1.ME.livesAt1 ,
22     pr-owl2:hasArgumentNumber 1
23
24 <!-- The second argument livesAt(person2) of equalTo(livesAt(person1), livesAt(
25     person2)) -->
26 Individual: MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1.MEA.arg2
27 Types:
28     pr-owl2:MExpressionArgument
29 Facts:
30     pr-owl2:isTypeOfArgumentIn MFrag.PersonalInformation.GIN.equalTo1.BME.
31         equalTo1.MEA.arg1 ,
32     pr-owl2:hasArgument MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1 .
33         ME.livesAt1.OVA.person1 ,
34     pr-owl2:typeOfMExpression RV.livesAt
35
36 <!-- The type of the second argument, which is the expression livesAt(person2)
37     -->
38 Individual: MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1.ME.livesAt1
39 Types:
40     pr-owl2:MExpression
41 Facts:
42     pr-owl2:isTypeOfArgumentIn MFrag.PersonalInformation.GIN.equalTo1.BME.
43         equalTo1.MEA.arg1 ,
44     pr-owl2:hasArgument MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1 .
45         ME.livesAt1.OVA.person1 ,
46     pr-owl2:typeOfMExpression RV.livesAt
47
48 <!-- The type of the first argument, which is the expression livesAt(person1)
49     -->
50 Individual: MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1.ME.livesAt1
51 Types:
52     pr-owl2:MExpression
53 Facts:
54     pr-owl2:isTypeOfArgumentIn MFrag.PersonalInformation.GIN.equalTo1.BME.
55         equalTo1.MEA.arg1 ,
56     pr-owl2:hasArgument MFrag.PersonalInformation.GIN.equalTo1.BME.equalTo1 .
57         ME.livesAt1.OVA.person1 ,
58     pr-owl2:typeOfMExpression RV.livesAt
59
60 <!-- The type of the second argument, which is the expression livesAt(person2)
61     -->

```

```

39 Individual: Mfrag. PersonalInformation .GIN. equalTo1 .BME. equalTo1 .ME. livesAt2
40   Types:
41     pr-owl2:MExpression
42   Facts:
43     pr-owl2:isTypeOfArgumentIn  Mfrag. PersonalInformation .GIN. equalTo1 .BME.
44       equalTo1 .MEA. arg2 ,
45     pr-owl2:hasArgument  Mfrag. PersonalInformation .GIN. equalTo1 .BME. equalTo1 .
46       ME. livesAt2 .OVA. person2 ,
47     pr-owl2:typeOfMExpression  RV. livesAt
48
49 <!-- The first and only argument person1 of the expression livesAt(person1) --
50   >
51 Individual: Mfrag. PersonalInformation .GIN. equalTo1 .BME. equalTo1 .ME. livesAt1 .
52   OVA. person1
53   Types:
54     pr-owl2:OrdinaryVariableArgument
55   Facts:
56     pr-owl2:isArgumentOf  Mfrag. PersonalInformation .GIN. equalTo1 .BME. equalTo1 .
57       ME. livesAt1 ,
58     pr-owl2:typeOfArgument  Mfrag. PersonalInformation .OV. person1 ,
59     pr-owl2:hasArgumentNumber  1
60
61 <!-- The first and only argument person2 of the expression livesAt(person2) --
62   >
63 Individual: Mfrag. PersonalInformation .GIN. equalTo1 .BME. equalTo1 .ME. livesAt2 .
64   OVA. person2
65   Types:
66     pr-owl2:OrdinaryVariableArgument
67   Facts:
68     pr-owl2:isArgumentOf  Mfrag. PersonalInformation .GIN. equalTo1 .BME. equalTo1 .
69       ME. livesAt2 ,
70     pr-owl2:typeOfArgument  Mfrag. PersonalInformation .OV. person2 ,
71     pr-owl2:hasArgumentNumber  1

```

Listing 4.17: Domain resident node `isRelated(person1, person2)`

```

1 Individual: Mfrag. PersonalInformation .DRN. isRelated
2   Types:
3     pr-owl2:DomainResidentNode
4   Facts:
5     pr-owl2:isResidentNodeIn  Mfrag. PersonalInformation ,
6     pr-owl2:hasParent  Mfrag. PersonalInformation .GIN. equalTo1 ,
7     pr-owl2:hasMExpression  Mfrag. PersonalInformation .DRN. isRelated .SME.
8       isRelated ,
9     pr-owl2:hasProbabilityDistribution  Mfrag. PersonalInformation .DRN.
10      isRelated .PT. dist1

```

Listing 4.17 presents the domain resident node with a local probability distribution

for the random variable `isRelated(person1, person2)`, conditioned on the input node `equalTo(livesAt(person1), livesAt(person2))`. The distribution is consistent with the idea that if two people live at the same address they are more likely to be related.

Listing 4.18 presents a simple MEBN expression for the resident node `isRelated(person1, person2)`. The type of formula for this MEBN expression is the random variable `isRelated`. Finally, it has two ordinary variable arguments, `person1` and `person2`.

Listing 4.18: Simple MEBN expression for domain resident node `isRelated(person1, person2)`

```

1 <!-- The simple expression isRelated(person1, person2) -->
2 Individual: Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated
3   Types:
4     pr-owl2:SimpleMExpression
5   Facts:
6     pr-owl2:typeOfMExpression RV. isRelated ,
7     pr-owl2:isMExpressionOf Mfrag. PersonalInformation .DRN. isRelated ,
8     pr-owl2:hasArgument Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated
9       .OVA. person1 ,
10      pr-owl2:hasArgument Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated
11        .OVA. person2
12
13 <!-- The first argument person1 of isRelated(person1, person2) -->
14 Individual: Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated .OVA. person1
15   Types:
16     pr-owl2:OrdinaryVariableArgument
17   Facts:
18     pr-owl2:isArgumentOf Mfrag. PersonalInformation .DRN. isRelated .SME.
19       isRelated ,
20     pr-owl2:typeOfArgument Mfrag. PersonalInformation .OV. person1 ,
21     pr-owl2:hasArgumentNumber 1
22
23 <!-- The second argument person2 of isRelated(person1, person2) -->
24 Individual: Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated .OVA. person2
25   Types:
26     pr-owl2:OrdinaryVariableArgument
27   Facts:
28     pr-owl2:isArgumentOf Mfrag. PersonalInformation .DRN. isRelated .SME.
29       isRelated ,
30     pr-owl2:typeOfArgument Mfrag. PersonalInformation .OV. person2 ,
31     pr-owl2:hasArgumentNumber 2

```

Finally, Listing 4.19 presents the declarative probability distribution for the domain resident node `isRelated(person1, person2)`. This distribution is consistent with the idea that if two people live at the same address, they are more likely to be related.

Listing 4.19: Declarative probability distribution for domain resident node `isRelated(person1, person2)`

```

1 Individual: MFragment.PortalInformation.DRN.isRelated.DD.dist1
2   Types:
3     pr-owl2:DeclarativeDistribution
4   Facts:
5     pr-owl2:isRepresentedAs "UnBBayes"^^xsd:string ,
6     pr-owl2:hasDeclaration
7       " if _any _person1 . person2 _have _(_livesAt ( person1 ) _=_livesAt ( person2 ) _)[
8         _true _=_0.9 ,
9         _false _=_0.1 ,
10        _absurd _=_0
11      ] _else _[
12        _true _=_0.001 ,
13        _false _=_0.999 ,
14        _absurd _=_0
15      ] "^^xsd:string

```

Finding Nodes

To define a finding in PR-OWL 2, we need to define one finding input node and one finding resident node within its finding MFragment. Figure 4.28 presents the finding MFragment and its nodes for defining the evidence that Bill has annual income of 75,000.00.

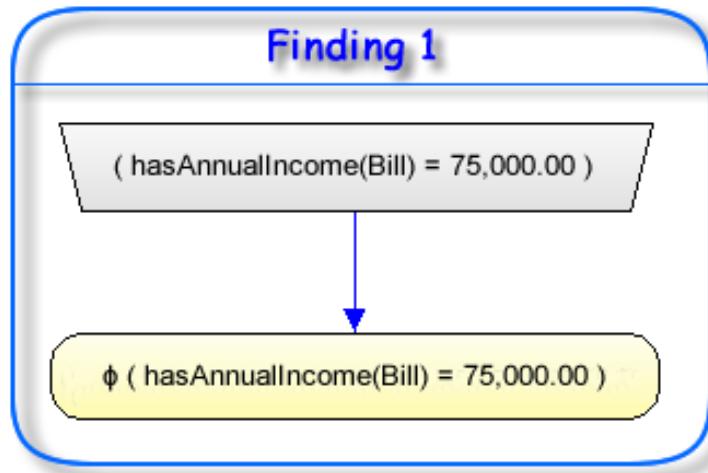


Figure 4.28: Nodes for representing the finding that Bill has annual income of 75,000.00.

Listing 4.20 presents the code for the finding MFragment in Figure 4.28, the `Finding1` MFragment. It has one finding resident node called `MFrag.Finding1.FRN.equalTo1` and one finding input node called `MFrag.Finding1.FIN.equalTo1`.

Listing 4.20: Finding MFragment for evidence `equalTo(hasAnnualIncome(Bill), 75,000.00)`

```

1 Individual: MFrag.Finding1
2   Types:
3     pr-owl2:FindingMFragment
4   Facts:
5     pr-owl2:isMFragmentOf MTTheory.FraudIdentificationInPublicProcurement ,
6     pr-owl2:hasInputNode MFrag.Finding1.FIN.equalTo1 ,
7     pr-owl2:hasResidentNode MFrag.Finding1.FRN.equalTo1

```

Listing 4.21 presents the finding input node (`MFrag.Finding1.FIN.equalTo1`) represented by the Boolean expression `equalTo(hasAnnualIncome(Bill), 75,000.00)` (`MFrag.Finding1.FIN.equalTo1.BME.equalTo1`), which is parent of the finding resident node `MFrag.Finding1.FRN.equalTo1`.

Listing 4.21: Finding input node for evidence `equalTo(hasAnnualIncome(Bill), 75,000.00)`

```

1 Individual: MFrag.Finding1.FIN.equalTo1
2   Types:
3     pr-owl2:FindingInputNode
4   Facts:
5     pr-owl2:isParentOf MFrag.Finding1.FRN.equalTo1 ,
6     pr-owl2:hasMExpression MFrag.Finding1.FIN.equalTo1.BME.equalTo1

```

Listing 4.22 presents how the Boolean expression `equalTo(hasAnnualIncome(Bill), 75,000.00)` for the finding input node `MFrag.Finding1.FIN.equalTo1` is defined.

Listing 4.22: Boolean expression of the finding input node for evidence `equalTo(hasAnnualIncome(Bill), 75,000.00)`

```

1 <!-- The Boolean expression equalTo(hasAnnualIncome(Bill), 75,000.00) -->
2 Individual: MFrag.Finding1.FIN.equalTo1.BME.equalTo1
3   Types:
4     pr-owl2:BooleanMExpression
5   Facts:
6     pr-owl2:typeOfMExpression pr-owl2:equalTo ,

```

```

7   pr-owl2:hasArgument Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.MA.arg1 ,
8   pr-owl2:isMExpressionOf Mfrag.Finding1.FIN.equalTo1 ,
9   pr-owl2:hasArgument Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.CA.float1
10
11 <!-- The first argument hasAnnualIncome(Bill) of equalTo(hasAnnualIncome(Bill)
12 , 75,000.00) -->
13 Individual: Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.MA.arg1
14   Types:
15     pr-owl2:MExpressionArgument
16   Facts:
17     pr-owl2:isArgumentOf Mfrag.Finding1.FIN.equalTo1.BME.equalTo1 ,
18     pr-owl2:typeOfArgument Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.ME.
19       hasAnnualIncome1 ,
20     pr-owl2:hasArgumentNumber 1
21
22 <!-- The type of the first argument, which is the expression hasAnnualIncome(
23 Bill) -->
24 Individual: Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.ME.hasAnnualIncome1
25   Types:
26     pr-owl2:MExpression
27   Facts:
28     pr-owl2:typeOfMExpression RV.hasAnnualIncome ,
29     pr-owl2:isTypeOfArgumentIn Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.MA.
30       arg1 ,
31     pr-owl2:hasArgument Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.ME.
32       hasAnnualIncome1.CA.person1
33
34 <!-- The first and only argument of the expression hasAnnualIncome(Bill) -->
35 Individual: Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.ME.hasAnnualIncome1.CA.
36   person1
37   Types:
38     pr-owl2:ConstantArgument
39   Facts:
40     pr-owl2:typeOfArgument Bill ,
41     pr-owl2:isArgumentOf Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.ME.
42       hasAnnualIncome1 ,
43     pr-owl2:hasArgumentNumber 1
44
45 <!-- The type of the first and only argument of the expression hasAnnualIncome(
46 (Bill), which is the person Bill -->
47 Individual: Bill
48   Types:
49     Person
50
51 <!-- The second argument 75,000.00 of equalTo(hasAnnualIncome(Bill),
52 75,000.00) -->
53 Individual: Mfrag.Finding1.FIN.equalTo1.BME.equalTo1.CA.float1
54   Types:
55     pr-owl2:ConstantArgument
56   Facts:
57     pr-owl2:isArgumentOf Mfrag.Finding1.FIN.equalTo1.BME.equalTo1 ,
58     pr-owl2:hasArgumentNumber 2 ,
59     pr-owl2:typeOfDataArgument 75000f

```

Listing 4.23 presents the finding resident node (`MFrag.Finding1.FRN.equalTo1`) represented by the Boolean expression `equalTo(hasAnnualIncome(Bill), 75,000.00)` (`MFrag.Finding1.FRN.equalTo1.BME.equalTo1`).

Listing 4.23: Finding resident node example

```

1 Individual: MFrag.Finding1.FRN.equalTo1
2   Types:
3     pr-owl2:FindingResidentNode
4   Facts:
5     pr-owl2:hasParent MFrag.Finding1.FIN.equalTo1 ,
6     pr-owl2:hasMExpression MFrag.Finding1.FRN.equalTo1.BME.equalTo1

```

Finally, Listing 4.24 presents how the finding resident node `MFrag.Finding1.FRN.equalTo1` is represented as the Boolean expression `equalTo(hasAnnualIncome(Bill), 75,000.00)`.

Listing 4.24: Boolean expression of the finding resident node for evidence `equalTo(hasAnnualIncome(Bill), 75,000.00)`

```

1 <!-- The Boolean expression equalTo(hasAnnualIncome(Bill), 75,000.00) -->
2 Individual: MFrag.Finding1.FRN.equalTo1.BME.equalTo1
3   Types:
4     pr-owl2:BooleanMExpression
5   Facts:
6     pr-owl2:typeOfMExpression pr-owl2:equalTo ,
7     pr-owl2:hasArgument MFrag.Finding1.FRN.equalTo1.BME.equalTo1.MA.arg1 ,
8     pr-owl2:isMExpressionOf MFrag.Finding1.FRN.equalTo1 ,
9     pr-owl2:hasArgument MFrag.Finding1.FRN.equalTo1.BME.equalTo1.CA.float1
10
11 <!-- The first argument hasAnnualIncome(Bill) of equalTo(hasAnnualIncome(Bill)
12   , 75,000.00) -->
13 Individual: MFrag.Finding1.FRN.equalTo1.BME.equalTo1.MA.arg1
14   Types:
15     pr-owl2:MExpressionArgument
16   Facts:
17     pr-owl2:isArgumentOf MFrag.Finding1.FRN.equalTo1.BME.equalTo1 ,
18     pr-owl2:typeOfArgument MFrag.Finding1.FRN.equalTo1.BME.equalTo1.ME.
19       hasAnnualIncome1 ,
20     pr-owl2:hasArgumentNumber 1
21
22 <!-- The type of the first argument, which is the expression hasAnnualIncome(
23   Bill) -->
24 Individual: MFrag.Finding1.FRN.equalTo1.BME.equalTo1.ME.hasAnnualIncome1
25   Types:
26     pr-owl2:MExpression
27   Facts:

```

```

25 pr-owl2:typeOfMExpression RV.hasAnnualIncome ,
26 pr-owl2:isTypeOfArgumentIn MFrag.Finding1.FRН. equalTo1.BME. equalTo1.MA.
27      arg1 ,
28 pr-owl2:hasArgument MFrag.Finding1.FRН. equalTo1.BME. equalTo1.ME.
29      hasAnnualIncome1.CA. person1
30
31 <!-- The first and only argument of the expression hasAnnualIncome(Bill) -->
32 Individual: MFrag.Finding1.FRН. equalTo1.BME. equalTo1.ME. hasAnnualIncome1.CA.
33      person1
34 Types:
35      pr-owl2:ConstantArgument
36 Facts:
37      pr-owl2:typeOfArgument Bill ,
38      pr-owl2:isArgumentOf MFrag.Finding1.FRН. equalTo1.BME. equalTo1.ME.
39      hasAnnualIncome1 ,
40      pr-owl2:hasArgumentNumber 1
41
42 <!-- The type of the first and only argument of the expression hasAnnualIncome
43 (Bill), which is the person Bill -->
44 Individual: Bill
45 Types:
46      Person
47
48 <!-- The second argument 75,000.00 of equalTo(hasAnnualIncome(Bill),
49      75,000.00) -->
50 Individual: MFrag.Finding1.FRН. equalTo1.BME. equalTo1.CA. float1
51 Types:
52      pr-owl2:ConstantArgument
53 Facts:
54      pr-owl2:isArgumentOf MFrag.Finding1.FRН. equalTo1.BME. equalTo1 ,
55      pr-owl2:hasArgumentNumber 2 ,
56      pr-owl2:typeOfDataArgument 75000 f

```

4.8 Types of Uncertainty Reasoning for the Semantic Web

Although there is no standardized query language for OWL[62], it is common to differentiate two types of reasoning: those associated with individuals; and those related to the ontological schema.

On the one hand, reasoning with individuals is usually associated with queries that ask for all individuals of a given class, or whether a given individual is of a given class. On the other hand, reasoning with the schema is usually associated with asking whether two classes are disjoint, classifying the ontology, or checking for global consistency.

From a logical perspective, the typical types of reasoning are:

1. Subsumption: to verify whether the knowledge base entails that $C \sqsubseteq D$;
2. Class equivalence: to verify whether the knowledge base entails that $C \equiv D$;
3. Class disjointness: to verify whether the knowledge base entails that $C \sqcap D \sqsubseteq \perp$;
4. Global consistency: to verify whether the knowledge base has at least one model;
5. Class consistency: to verify that the knowledge base does not entail that class $C \sqsubseteq \perp$;
6. Instance checking: to verify that the knowledge base entails that an individual a belongs to a class C , *i.e.*, $C(a)$;
7. Instance retrieval: to retrieve all individuals of a given class.

Fortunately, all these different reasoning problems can be reduced to the one of satisfiability, *i.e.*, whether the knowledge base has at least one model. This is a well-known problem and several algorithms have been proposed with varying degrees of complexity, depending on the underlying logic behind the ontological language being used.

In other words, the community is already familiar with the types of reasoning available for the semantic web when using logic-based languages like OWL. The question now becomes: “What types of reasoning can one expect from probabilistic languages for the semantic web?”

Poole *et al.* [110] emphasizes that it is not clear how to match the formalization of random variables from probabilistic theories with the concepts of individuals, classes and properties from current ontological languages like OWL. However, Poole *et al.* [110] says that “We can reconcile these views by having properties of individuals correspond to random variables.” This is the approach used in this work to integrate MEBN logic and OWL language.

Also according to Poole *et al.* [110], when integrating probability theories and ontological languages three types of uncertainty reasoning are expected:

- Existential uncertainty: what is the probability that an individual actually exists?

- Type uncertainty: what is the probability that an individual belongs to a given class?
- Property value uncertainty: what is the probability that an individual has a property with a given value?

Although there is no built-in random variable to define existential uncertainty in PR-OWL, it is easy to define such a random variable. For instance, Costa [27] presents the random variable `Exists(st)` to reason about whether its argument is an actual starship. In his example, he tries to verify whether a sensor report corresponds to one of the existing starships, to a new one, or if this was just a spurious sensor report, and this starship does not exist at all.

Poole *et al.* [110] also states that type uncertainty can be reduced to property value uncertainty. This is exactly the approach described in Section 4.6, where we use the built-in `isA(resource, class)` random variable to define the uncertainty of the RDF `type` property. In fact, the careful reader might notice that we have also reduced the existential uncertainty to property value uncertainty. Reasoning about whether an individual exists became identifying whether the property `exists(individual)` has value true for the individual in question.

These types of uncertainty described by Poole *et al.* [110] are not the only ones. For instance, Costa [27] talks about identity uncertainty and association uncertainty (also known as reference uncertainty [47]). These too can be easily reduced to property value uncertainty. Therefore, property value uncertainty plays a crucial role in probabilistic first-order language reasoning, just as satisfiability does for logical reasoning. Note that uncertainty reasoning augments logical reasoning rather than replacing it. That is, the types of reasoning identified above as typical for logical knowledge bases are required for probabilistic knowledge bases, along with the types of uncertainty reasoning described by Poole *et al.* [110].

Finally, it is important to notice that PR-OWL 2 besides integrating well with the web ontology language (OWL) by mapping random variables to properties, it supports all major uncertainty reasoning expected from a probabilistic first-order language.

Chapter 5: Uncertainty Modeling Process for Semantic Technologies (UMP-ST)

As explained in Chapter 1, probabilistic ontologies can be used to represent experts' knowledge in an automated system in order to overcome the information overload problem. However, one major problem is that probabilistic ontologies are complex and hard to model. It is challenging enough to design models that use only logic or only uncertainty; combining the two poses an even greater challenge. In fact, in the past few years I have received a number of e-mails from researchers all around the world asking for some information and/or literature on how to build probabilistic ontologies. The problem is that there is no methodology in the literature related to probabilistic ontology engineering.

Although there is now substantial literature about what PR-OWL is [27, 29, 31], how to implement it [23, 20, 19, 26], and where it can be used [30, 32, 33, 77, 79, 80], little has been written about how to model a probabilistic ontology.

This lack of methodology is not only associated with PR-OWL. Other languages that use probabilistic methods for representing uncertainty on the SW have been advancing in areas like inference [12, 122], learning [36, 86], and applications [14, 120, 87, 13, 41]. Examples of such languages include OntoBayes [136], BayesOWL [37], and probabilistic extensions of SHIF(**D**) and SHOIN(**D**) [85], and Markov Logic Networks (MLN). Despite this proliferation of languages and methods, little has been written about how to build such models.

Therefore, in this Chapter I will describe an approach for modeling a probabilistic ontology and using it for plausible reasoning in applications that use Semantic Technologies.

The Uncertainty Reasoning Process for Semantic Technologies (URP-ST)¹ presented

¹In [22] I present this process as the modeling process. However, this is actually more than just modeling. This process represents the sequence of phases necessary in order to achieve the capability of plausible reasoning with semantic technologies. Therefore, I have changed the name of this process to Uncertainty Reasoning Process for Semantic Technologies (URP-ST).

in Figure 5.1 is divided into three steps: First we have to model the domain (T-Box²), then we need to populate the model with data (A-Box³), and finally we can use both the model (T-Box) and the data available (A-Box), *i.e.*, the KB, for reasoning. In other words, in order to be able to reason with uncertainty, first we need a model, which describes how the different concepts in our ontology interact under uncertainty by knowing which evidence supports which hypothesis, etc. Once there is a model available, it needs to be populated with the data available before it is able to do any reasoning. Finally, with the model and with the data available, it is possible to present the inference engine with queries for that domain, like `isA(person1, Terrorist)`. Notice that unlike standard ontology reasoning systems that return true only if that person is known to be a terrorist for sure, the probabilistic ontology reasoning system will return the likelihood that the person is a terrorist, for instance $P(\text{isA}(\text{person1}, \text{Terrorist}) = \text{true}) = 75\%$.

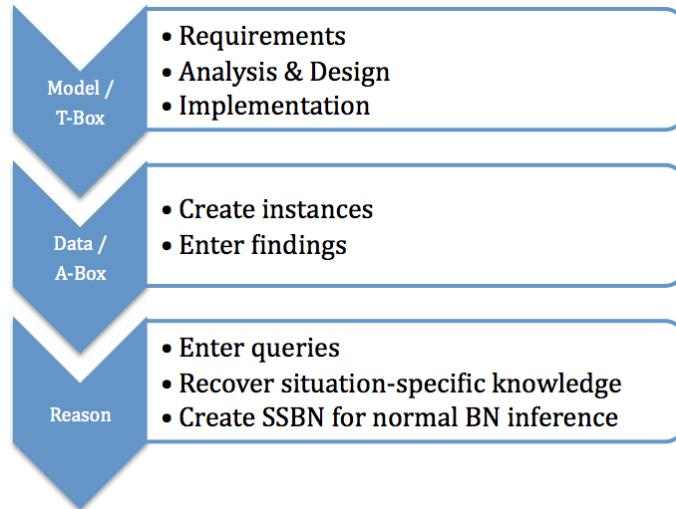


Figure 5.1: Uncertainty Reasoning Process for ST (URP-ST).

²T-Box statements describe the part of the KB that defines terms of a controlled vocabulary, for example, a set of classes and properties

³A-Box are statements about the vocabulary defined by the T-Box, for example, instances of classes. T-Box and A-Box together form the KB.

Now I focus in detail on the modeling phase of the URP-ST. I call this phase the Uncertainty Modeling Process for Semantic Technologies (UMP-ST). The UMP-ST consists of four major disciplines: Requirements, Analysis & Design, Implementation, and Test. These terms are borrowed from the Unified Process (UP)⁴ [68] with some modifications to reflect our domain of ontology modeling instead of software development process. The methodology described here is also consistent with the Bayesian network modeling methodology described by [72] and [81].

Figure 5.2 depicts the intensity of each discipline during the UMP-ST⁵. Like the UP, UMP-ST is iterative and incremental. The basic idea behind iterative enhancement is to model our domain incrementally, allowing the modeler to take advantage of what was being learned during the modeling of earlier, incremental, deliverable versions of the model. Learning comes from discovering new rules, entities, and relations that were not obvious previously, which can give rise to new questions and evidence that might help us achieve our previously defined goal as well as give rise to new goals. Some times it is possible to test some of the rules defined during the Analysis & Design stage even before having implemented it. This is usually done by creating simple probabilistic models to evaluate whether the model will behave as expected before creating the more complex first-order logic probabilistic models. That is why in the first iteration (I1) of the Inception phase we have some testing happening before the implementation started.

⁴Although the most common instance of UP is the Rational Unified Process (RUP) [74], there are alternatives, like the Open Unified Process (OpenUP) [9].

⁵In [22] I present this methodology as UMP for the Semantic Web. However, this methodology is not restricted to the SW. Any application that uses semantic technologies can benefit from it, even if it is not designed to be used on the Web. Therefore, I decided to change the name to UMP for Semantic Technologies.

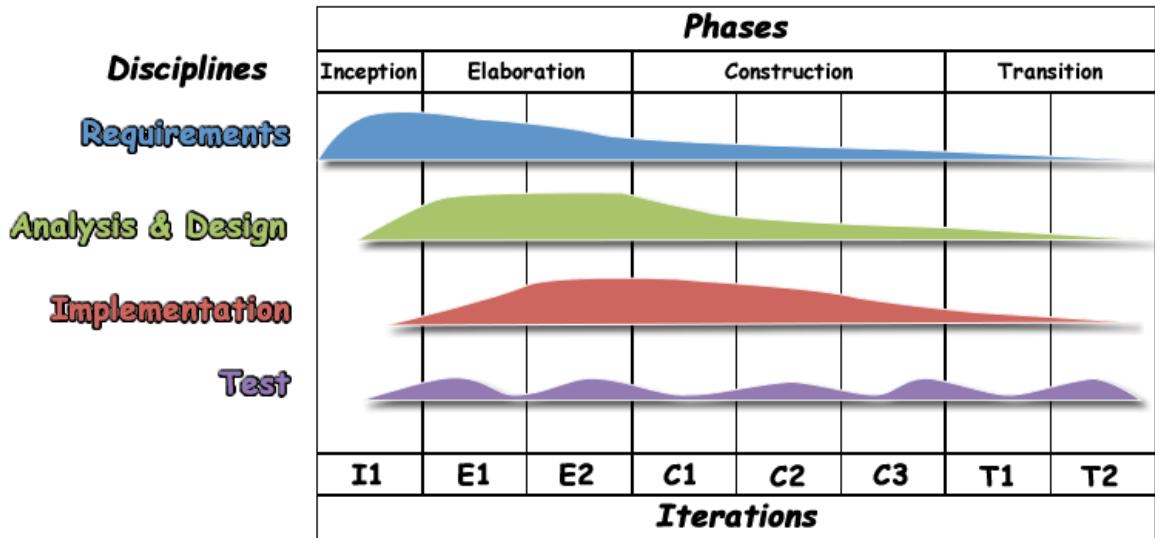


Figure 5.2: Uncertainty Modeling Process for Semantic Technologies (UMP-ST).

Figure 5.3 presents the Probabilistic Ontology Modeling Cycle (POMC). This cycle depicts the major activities or concepts in each discipline, how they usually interact, and the natural order in which they occur. However, as described previously, this is not the same as the waterfall model (see [114] for information about the waterfall model). *I.e.*, it is not necessary to go through implementation to be able to test the model. Besides that, the interactions between the disciplines are not restricted to the arrows presented. In fact, it is possible to have interactions between any pair of disciplines. For instance, it is not uncommon to discover a problem in the rules defined in the Analysis & Design discipline during the activities in the Test discipline. In other words, although, the arrow just shows interaction between the Test and Requirement disciplines, it is possible to go directly from Test to Analysis & Design.



Figure 5.3: Probabilistic Ontology Modeling Cycle (POMC) - Requirements (Goals), Analysis & Design (Entities, Rules, and Group), Implementation (Mapping and LPD), and Test (Evaluation).

In Figure 5.3 the Requirements discipline (Goals circle in blue) defines the goals that must be achieved by reasoning with the semantics provided by our model. The Analysis & Design discipline describes classes of entities, their attributes, how they relate, and what rules apply to them in our domain (Entities, Rules, and Group circles in green). This definition is independent of the language used to implement the model. The Implementation discipline maps our design to a specific language that allows uncertainty in ST, which in this

case is PR-OWL (Mapping and LPD circles in red). Finally, the Test discipline is responsible for evaluating if the model developed during the Implementation discipline is behaving as expected from the rules defined during Analysis & Design and if they achieve the goals elicited during the Requirements discipline (Evaluation circle in purple). As explained before, it is possible to test some rules and assumptions even before the implementation. This is a crucial step to mitigate risk by identifying problems before wasting time in developing an inappropriate complex model.

The following sections illustrate the UMP-ST process and the POMC cycle through a case study in procurement fraud detection and prevention and a case study in maritime domain awareness. The URP-ST is also demonstrated by the use of UnBBayes to implement the model, to populate the KB, and to perform plausible reasoning.

On the one hand, Section 5.1 will focus on presenting in detail the activities that must be executed in each discipline in the POMC cycle. On the other hand, Section 5.2 will focus on presenting how the model evolves through time with every new iteration.

The objective of the first is to present as much detail as possible on the steps necessary to model a probabilistic ontology using the POMC cycle. The objective of the second is to show that the UMP-ST process provides a useful approach for allowing the natural evolution of the model through different iterations.

5.1 Probabilistic Ontology for Procurement Fraud Detection and Prevention in Brazil

A major source of corruption is the procurement process. Although laws attempt to ensure a competitive and fair process, perpetrators find ways to turn the process to their advantage while appearing to be legitimate. This is why a specialist has didactically structured the different kinds of procurement frauds the Brazilian Office of the Comptroller General (CGU) has dealt with in past years.

These different fraud types are characterized by criteria, such as business owners who

work as a front for the company, use of accounting indices that are not common practice, etc. Indicators have been established to help identify cases of each of these fraud types. For instance, one principle that must be followed in public procurement is that of competition. Every public procurement should establish minimum requisites necessary to guarantee the execution of the contract in order to maximize the number of participating bidders. Nevertheless, it is common to have a fake competition when different bidders are, in fact, owned by the same person. This is usually done by having someone as a front for the enterprise, which is often someone with little or no education.

The ultimate goal of this case study is to structure the specialist knowledge in a way that an automated system can reason with the evidence in a manner similar to the specialist. Such an automated system is intended to support specialists and to help train new specialists, but not to replace them. Initially, a few simple criteria were selected as a proof of concept. Nevertheless, it is shown that the model can be incrementally updated to incorporate new criteria. In this process, it becomes clear that a number of different sources must be consulted to come up with the necessary indicators to create new and useful knowledge for decision makers about the procurements.

Figure 5.4 presents an overview of the procurement fraud detection process. The data for our case study represent several requests for proposal and auctions that are issued by the Federal, State and Municipal Offices (Public Notices - Data). The idea is that the analysts who work at CGU, already making audits and inspections, accomplish the collection of information through questionnaires that can specifically be created for the collecting of indicators for the selected criteria (Information Gathering). These questionnaires can be created using a system that is already in production at CGU. Once they are answered the necessary information is going to be available (DB - Information). Hence, UnBBayes, using the probabilistic ontology designed by experts (Design - UnBBayes), will be able to collect these millions of items of information and transform them into dozens or hundreds of items of knowledge. This will be achieved through logic and probabilistic inference. For instance, procurement announcements, contracts, reports, etc. - a huge amount of data - are analyzed

allowing the gathering of relevant relations and properties - a large amount of information. Then, these relevant relations and properties are used to draw some conclusions about possible irregularities - a smaller number of items of knowledge (Inference - Knowledge). This knowledge can be filtered so that only the procurements that show a probability higher than a threshold, *e.g.* 20%, are automatically forwarded to the responsible department along with the inferences about potential fraud and the supporting evidence (Report for Decision Makers).



Figure 5.4: Procurement fraud detection overview.

5.1.1 Requirements

The objective of the requirements discipline is to define the objectives that must be achieved by creating a computable representation of domain semantics and reasoning with it. For this discipline, it is important to define the questions that the model is expected to answer

(*i.e.*, the queries to be posed to the system being designed). For each question, a set of information that might help answer the question (evidence) must be defined.

There are basically two types of requirements: functional and non functional [134, 124]. The requirements just described above are called functional requirements. Functional requirements are statements related to what the system should provide, what features it should have, how it should behave, etc. In our case, functional requirements relate to the goals, queries, and evidence that pertain to our domain of reasoning. Non functional requirements on the other hand represent constraints on the system as a whole. For instance, in our use case a non functional requirement could be that the query has to be answered in less than a minute. Another example is that the posterior probability given as an answer to a given query has to be either exact or an approximation with an error bound of .5%.

Since it is easier and more straightforward to define non functional requirements, which define time constraints, error bounds, etc., we will focus on describing how to come up with the functional requirements in our use case.

In order to understand the requirements for the procurement fraud detection and prevention model, we first have to explain some of the problems encountered when dealing with public procurements.

One of the principles established by the Law N 8,666/93 is equality among the bidders. This principle prohibits the procurement agent from discriminating among potential suppliers. However, if the procurement agent is related to the bidder, he/she might feed information or define new requirements for the procurement in a way that favors the bidder.

Another principle that must be followed in public procurement is that of competition. Every public procurement should establish minimum requisites necessary to guarantee the execution of the contract in order to maximize the number of participating bidders. Nevertheless, it is common to have a fake competition when different bidders are, in fact, owned by the same person. This is usually done by having someone as a front for the enterprise, which is often someone with little or no education. Another common tactic is to set up front enterprises owned by relatives of the owner of the enterprise committing fraud.

According to [98] participating in a public procurement can be very expensive and time consuming. Thus, some firms are unwilling to take part in a process that is not guaranteed to achieve favorable results. Since this diminishes the number of enterprises participating in the procurement, collusion among the bidders is more likely to happen. What happens in Brazil is that a small group of firms regularly participate in procurements of certain goods and services. When this happens, the competitors in a public procurement take turns winning the contracts. They stipulate the winning bid, and all other firms bid above that price. There is no competition, and the government pays a higher price for the contract. Although collusion is not an easy thing to prove, it is reasonable to assume that collusion is enabled by some kind of relationship between the enterprises.

All firms in Brazil have a registration number, called CGC, which stands for General List of Contributors. When a firm is suspended from procuring with the public administration, its CGC number is used to inform all other public agencies that this firm should not participate in public procurements. However, the firm can simply close its business and open a new one using a different CGC. Thus the firm that should not be able to participate in public procurements is now allowed, since it now has a different number associated to it. Unfortunately, the Commercial Code permits this change of CGC number.

One other problem is that public procurement is quite complex and may involve large sums of money. Therefore, the members that form the committee of the procurement must not only be prepared, but also have a clean history (no criminal nor administrative conviction) in order to maximize morality, one of the ethical principles that federal, state, municipal and district government should all adopt.

Having explained that, in our fraud detection and prevention in the procurements domain we have the following set of goals/queries/evidences:

1. Identify whether a given procurement should be inspected and/or audited (*i.e.* evidence suggests further analysis is needed);
 - (a) Is there any relation between the committee and the enterprises that participated

in the procurement?

i. Look for member and responsible person of an enterprise who are related (mother, father, brother, or sister);

ii. Look for member and responsible person of an enterprise who live at the same address.

(b) Is the responsible person of the winner enterprise of the procurement a front?

i. Look for value of the contract related to this procurement;

ii. Look for his/her education degree;

iii. Look for his/her annual income.

(c) Was the responsible person of the winner enterprise of the procurement responsible for an enterprise that has been suspended from procuring with the public administration?

i. Look for this information in the General List of Contributors (CGC) database.

(d) Was competition compromised?

i. Look for bidders who are related (mother, father, brother, or sister).

2. Identify whether the committee of a given procurement should be changed.

(a) Is there any member of committee who does not have a clean history?

i. Look for criminal history;

ii. Look for administrative investigations.

(b) Is there any relation between members of the committee and the enterprises that participated in previous procurements?

i. Look for member and responsible person of an enterprise who are relatives (mother, father, brother, or sister);

ii. Look for member and responsible person of an enterprise who live at the same address.

Another important aspect of the Requirements discipline is defining traceability of requirements. Gotel and Finkelstein [51] define requirements traceability as:

Requirements traceability refers to the ability to describe and follow the life of a requirement, in both forwards and backwards direction.

A common tool for defining requirements traceability is the specification tree, which is the arrangement of requirements in such a way that each requirement is linked to its “parent” requirement in the higher specification. This is exactly the way we have defined the requirements for our procurement model. Every evidence is linked to its higher level query, which is linked to its higher level goal. Here we are not only defining the requirements, but also defining their traceability.

However, requirements traceability (RT) is not only about defining links between requirements. In fact, RT also provides the link between work products of other disciplines, like the rules in the Analysis & Design and MFrags in the Implementation, and the goals, queries, and evidence elicited in the Requirements discipline. This kind of link makes RT specially useful for validation and management of change.

This kind of link between work products of different disciplines is typically done via a Requirements Traceability Matrix (RTM) [134, 124]. Table 5.1 presents a RTM with the traceability between the requirements defined in this Section for the fraud detection model. Notice that this matrix represents exactly the same thing as the specification tree defined previously. However, when mapping the work product of other disciplines to the requirements, in most cases, it will not be possible to use a specification tree, but it will always be possible to use RTM.

5.1.2 Analysis & Design

Once we have defined our goals and described how to achieve them, it is time to start modeling the entities, their attributes, relationships, and rules to make that happen. This is the purpose of the Analysis & Design discipline.

Table 5.1: Requirements Traceability Matrix for the requirements of the fraud detection model.

ID	1	1a	1ai	1aai	1b	1bi	1bii	1biii	1c	1ci	1d	1di	2	2a	2ai	2aai	2b	2bi	2bii
1	X																		
1a	X	X																	
1ai	X	X	X																
1aai	X	X		X															
1b	X				X														
1bi	X					X	X												
1bii	X						X												
1biii	X							X											
1c	X								X										
1ci	X									X	X								
1d	X										X								
1di	X											X	X						
2													X						
2a													X	X					
2ai													X	X	X				
2aai													X	X	X	X			
2b													X				X		
2bi													X			X	X		
2bii													X			X		X	

The major objective of this discipline is to define the semantics of our model. In fact, most of our semantics can be defined in normal ontologies, including the deterministic rules that the concepts described in our model must obey. Since there are whole books describing how to design such ontologies, and our main concern is on the uncertain part of the ontology, we will not cover these methods in this Section. For more information see [7, 50, 101, 102].

Nevertheless, we do need a starting point in order to design our probabilistic ontology. As a matter of fact, one good way to start modeling these properties is to use UML as described in Section 2.1. However, as we have seen, UML does not support complex rule definitions. So we will just document them separately to remind us of the rules that must be described when implementing our model in PR-OWL.

Figure 5.5 depicts a simplified design of our domain requirements. A **Person** has a **name**, a **mother** and a **father** (also **Person**). Every **Person** has a unique identification that in Brazil is called **CPF**. A **Person** also has an **Education** and **livesAt** a certain **Address**. In addition, everyone is obliged to file his/her **TaxInfo** every year, including his/her **annual-Income**. These entities can be grouped as **Personal Information**. A **PublicServant** is a **Person** who **worksFor** a **PublicAgency**, which is a Government Agency. Every public **Procurement** is owed by a **PublicAgency**, has a committee formed by a group of **Public-Servants**, and has a group of **participants**, which are **Enterprises**. One of these will be the **winner** of the **Procurement**. Eventually, the **winner** of the **Procurement** will receive a **Contract** of some **value** with the **PublicAgency** owner of the **Procurement**. The entities just described can be grouped as **Procurement Information**. Every **Enterprise** has at least one **Person** that is **responsible** for its legal acts.

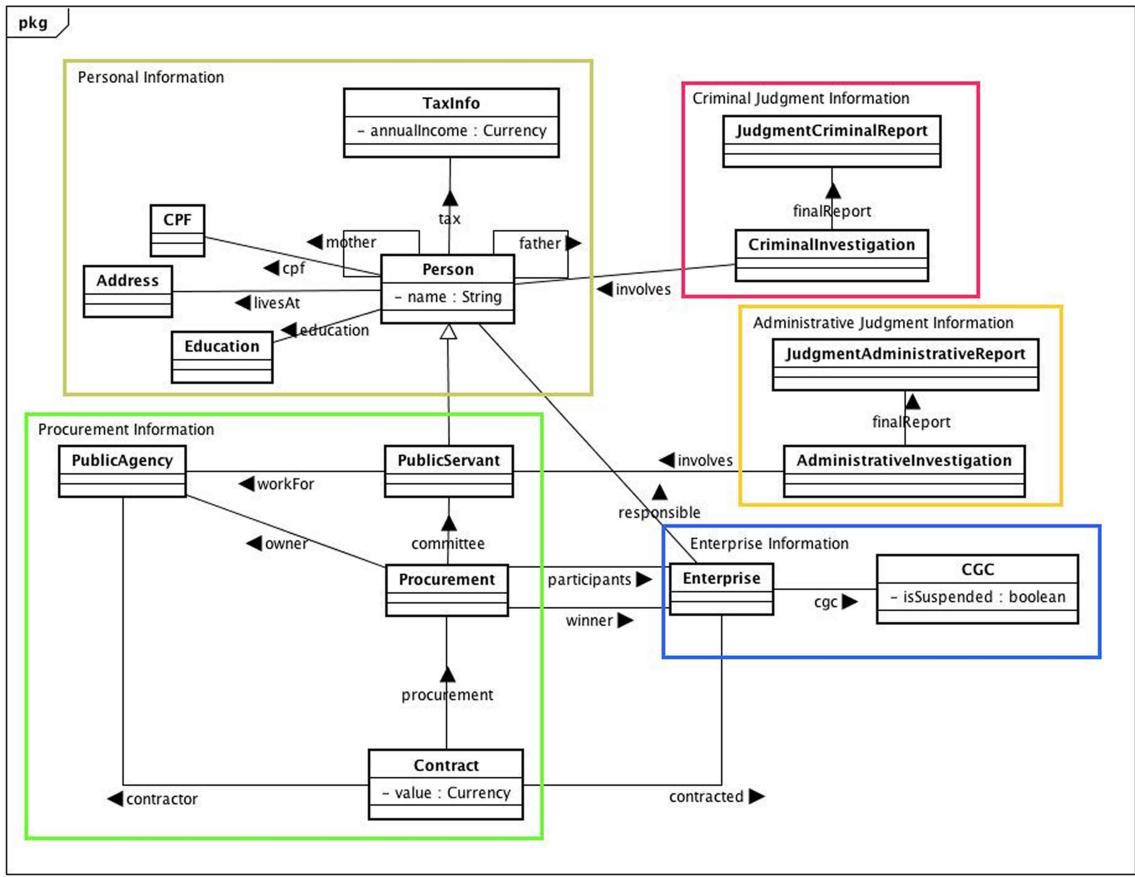


Figure 5.5: Entities, their attributes, and relations for the procurement model.

An **Enterprise** also has an identification number, the General List of Contributors **CGC**, which can be used to inform that this **Enterprise** is suspended from procuring with the public administration, **isSuspended**. These are grouped as the **Enterprise Information**. We also have **AdministrativeInvestigation**, which has information about investigations that involves one or more **PublicServer**. Its **finalReport**, the **JudgmentAdministrativeReport**, contains information about the penalty applied, if any. These entities form the **Administrative Judgment Information**. Finally we have the **Criminal Judgment Information** group that describes the **CriminalInvestigation** that involves a **Person**, with its **finalReport**, the **JudgmentCriminalReport**, which has information about the verdict.

Table 5.2: Requirements Traceability Matrix for the rules of the fraud detection model.

ID	1	1a	1ai	1aii	1b	1bi	1bii	1biii	1c	1ci	1d	1di	2	2a	2ai	2aii	2b	2bi	2bii
1	X	X	X																
2	X	X		X							X								
3	X			X	X	X					X								
4	X									X	X								
5	X										X	X							
6	X	X	X	X	X	X	X	X	X	X	X	X							
7	X												X	X	X	X			
8	X												X	X	X	X	X	X	
9	X													X	X	X	X	X	

Besides the cardinality and uniqueness rules defined in the explanation above about the entities depicted in Figure 5.5, the probabilistic rules for our model include:

1. If a member of the committee has a relative (mother, father, brother, or sister) responsible for a bidder in the procurement, then it is more likely that a relationship exists between the committee and the enterprises, which inhibits competition.
2. If a member of the committee lives at the same address as a person responsible for a bidder in the procurement, then it is more likely that a relationship exists between the committee and the enterprises, which lowers competition.
3. If a contract of high value related to a procurement has a responsible person of the winner enterprise with low education or low annual income, then this person is likely to be a front for the firm, which lowers competition.
4. If the responsible person of the winner enterprise is also responsible for another enterprise that has its CGC suspended for procuring with the public administration, then this procurement is more likely to need further investigation.
5. If the responsible people for the bidders in the procurement are related to each other, then a competition is more likely to have been compromised.
6. If 1, 2, 3, or 5, then the procurement is more likely to require further investigation.
7. If a member of the committee has been convicted of a crime or has been penalized administratively, then he/she does not have a clean history. If he/she was recently investigated, then it is likely that he/she does not have a clean history.
8. If the relation defined in 1 and 2 is found in previous procurements, then it is more likely that there will be a relation between this committee and future bidders.
9. If 7 or 8, then it is more likely that the committee needs to be changed.

Once we have our rules defined, it is important to keep track of their traceability to the requirements. Although this is a step of the Requirements discipline, we will present it here.

In fact, when completing every discipline it is important to go back to the Requirements discipline to expand the RTM matrix.

Table 5.2 presents the traceability between the rules defined in the Analysis & Design stage and the goals, queries, and evidence defined in the Requirements stage. *I.e.*, this mapping defines which requirements the rules are realizing.

5.1.3 Implementation

Once we have finished our Analysis & Design, it is time to start implementing our model in a specific language. This Section describes how to model procurement fraud detection and prevention in PR-OWL using UnBBayes.

The first thing to do is to start mapping the entities, their attributes, and relations to PR-OWL, which uses essentially MEBN terms. This discipline is different from the previous ones, since it depends on the language/formalism being used. In this Section I will highlight the difference between implementing the fraud detection probabilistic ontology using PR-OWL 1 and PR-OWL 2.

PR-OWL 1, although with a few limitations, already has a mature implementation in UnBBayes (the first version was made publicly available in February 2008). PR-OWL 2 on the other hand is still under development [88] and the current working version has a lot of limitations and is just a proof-of-concept⁶. Therefore, the fraud detection probabilistic ontology will not be fully implemented in PR-OWL 2, but it will be implemented in PR-OWL 1. Nevertheless, once the final version of PR-OWL 2 is available it should be straightforward to migrate this PO to PR-OWL 2. Notice that the main objective of this Chapter is to describe the UMP-ST process and to highlight the differences between PR-OWL 1 and PR-OWL 2.

In PR-OWL 1, it is often a good idea to start mapping the entities. There is no need to map all entities in our model to an entity in PR-OWL. In fact, in our model we will make many simplifications. One of them is due to a limitation in UnBBayes current version, which

⁶PR-OWL 2 is being developed by the Group of Artificial Intelligence (GIA) at the University of Brasília, Brazil.

is the lack of support for a type hierarchy. Therefore, we will not have the `PublicServant` entity and we will assume that a `Person` might work for a `PublicAgency`. We will also assume that every `Person` and `Enterprise` in our KB is uniquely identified by its name, so we will not consider, in this simplified example, the `CPF` and `CGC` entities. Figure 5.6(a) presents the entities implemented in our PR-OWL ontology using UnBBayes. For more details about defining entities in UnBBayes see [20].

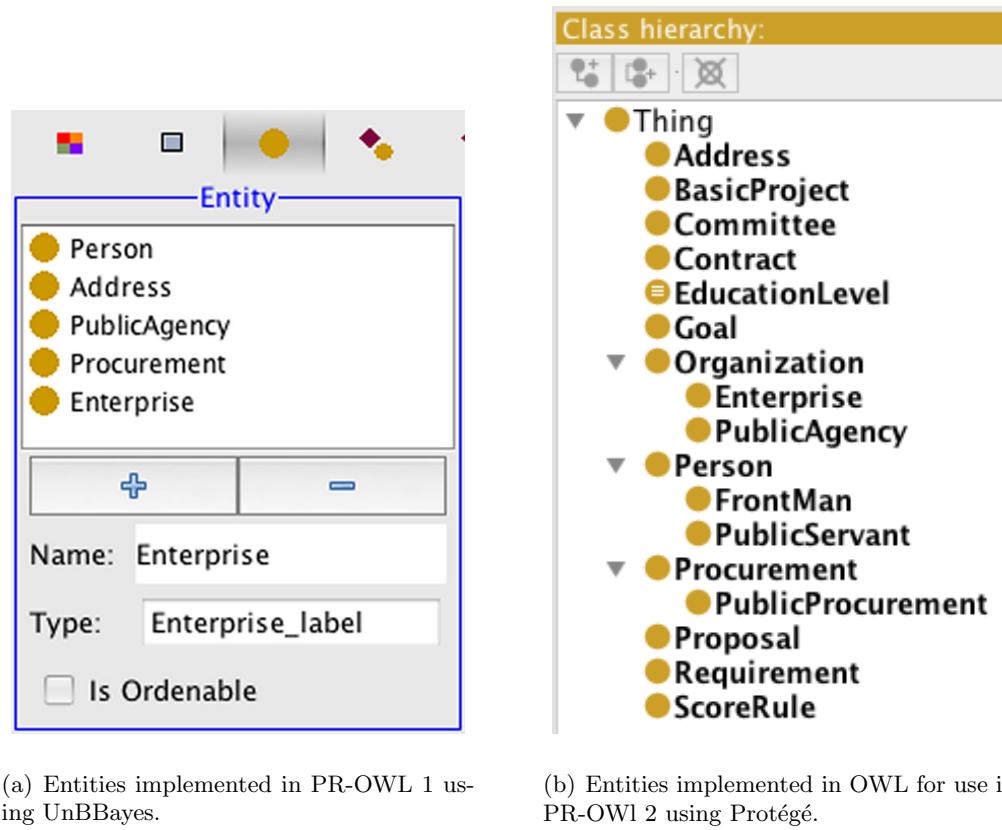


Figure 5.6: Entities for the procurement domain.

In PR-OWL 2, on the other hand, it is not necessary to map these entities. In fact, the entities are defined as classes in a regular ontology using OWL. Then PR-OWL 2 simply makes use of them. As previously explained, it is not the objective of the UMP-ST process to explain how to design standard deterministic ontologies. However, the Analysis & Design

discipline helps with a starting point for defining this ontology. The class hierarchy presented in Figure 5.6(b) was derived from the UML diagram created during the Analysis & Design stage presented in Figure 5.5.

Once we have our entities defined, we consider characteristics that may be uncertain. Uncertainty is represented in MEBN by defining random variables (RVs). On the one hand, to define a RV in PR-OWL 1 using UnBBayes, we first define its home MFragment. Grouping the RVs into MFrags is done by examining the grouping created during Analysis & Design. On the other hand, in PR-OWL 2 RVs are independent of MFragment and are defined globally by defining its arguments, mapping to OWL, and default distributions.

Typically, a RV represents an attribute or a relation from our designed model in Analysis & Design. For instance, the RV `livesAt(person)` maps the relation `livesAt` in our designed model. As it is a functional relation, `livesAt` relates a `Person` to an `Address`. Hence, the possible values (or states) of this RV are instances of `Address`.

It is important to notice that although we followed the best practice of having the same domain and range on both OWL terms (*e.g.* `livesAt`) and PR-OWL 1 random variables (*e.g.* `livesAt(person)`), there is nothing in the language that guarantees these manual mappings will be kept the same throughout the life cycle of the model. Moreover, since there is no formal link between these terms, it is impossible for reasoners to identify that these terms are even linked. At best, it could only “guess” they are the same, since they have similar syntax (*e.g.* predicate `livesAt` has a similar name to the random variable `livesAt(person)`), which is, at best, contradictory for a language that is designed to convey semantics of terms and relations.

Chapter 4 described how PR-OWL 2 formalizes the mapping between RVs and OWL properties. In the proof-of-concept PR-OWL 2 plugin for UnBBayes, from now on called PR-OWL 2 plugin [88], a RV is automatically created and its mapping automatically defined by dragging the OWL property and dropping it in the MFragment where it will be used as a resident node, as shown in Figure 5.7.

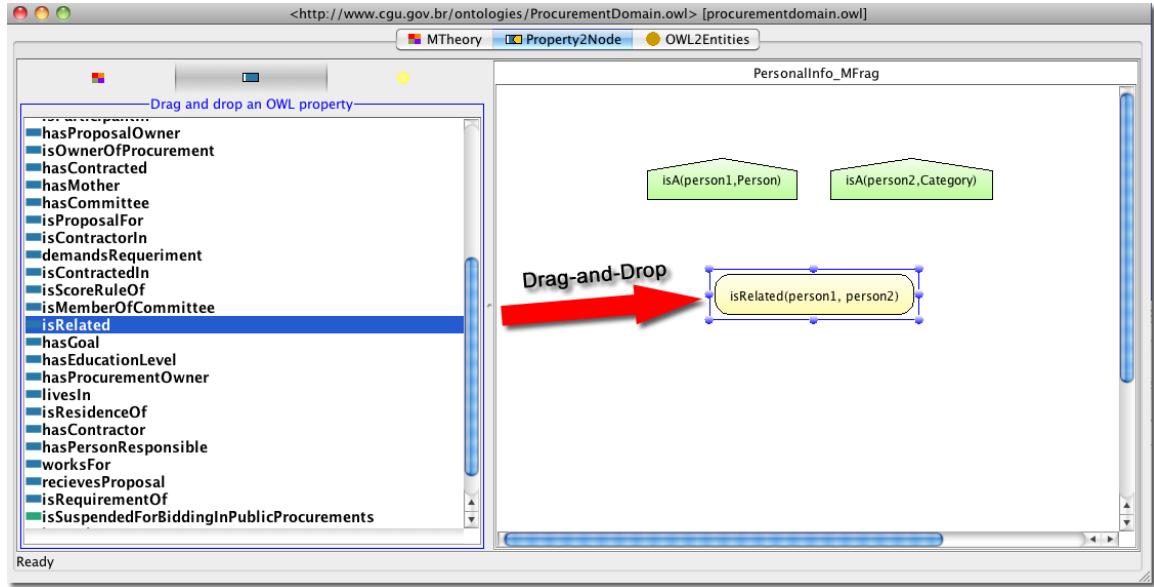


Figure 5.7: Creating a RV in PR-OWL 2 plugin from its OWL property by drag-and-drop.

We can also avoid explicitly representing some entities, by simply defining discrete outputs. In our implementation, we only need to know the education level of a **Person**, which is either `noEducation`, `middleSchool`, `highSchool`, `undergraduate`, or `graduate`. These are the states of the RV `hasEducationLevel(person)`, therefore, in PR-OWL 1, there is no need to define the entity `EducationLevel`, since no actual mapping will exist between the categorical RV and the OWL property `hasEducationLevel`. However, in PR-OWL 2, in order to represent categorical values, we would create a class `EducationLevel` with the `oneOf` construct from OWL. This construct allows us to define a set of predefined possible values for that class, which is exactly what we need.

Because the current version of UnBBayes-MEBN does not support continuous RVs, we must define a discretization for numerical attributes. For example, the attribute `value` of the `Contract` entity from our designed model is continuous, since it represents some `float` value in a specific `Currency`. However, we can discretize it by defining common intervals, as `lower than 10,000.00`, `between 10,000.01 and 100,000.00`, `between 100,000.01 and 500,000.00`, `between 500,000.01 and 1,000,000.00`, and `greater`

than 1,000,000.01, which will be the states of the resident node `valueOf(procurement)`. This is the case for both implementations of PR-OWL 1 and PR-OWL 2 in UnBBayes. The difference is that in future versions of UnBBayes, which will support continuous RVs, PR-OWL 1 will not be able to use data types such as `float`, while PR-OWL 2 will, since the latter uses OWL's types instead of defining its own types as the former does.

Once all resident RVs are created, their relations can be defined by analyzing dependence between nodes. One good way to look for dependence is by looking at the rules defined in our model. For instance, rule 3 indicates that there is a dependence between `valueOf(procurement)`, `hasEducationLevel(person)`, and `isFront(person, enterprise)`.

The MFrags implemented in order to address all the rules defined in the Analysis & Design are:

1. Personal Information
2. Procurement Information
3. Enterprise Information
4. Front of Enterprise
5. Exists Front in Enterprise
6. Related Participant Enterprises
7. Member Related to Participant
8. Competition Compromised
9. Owns Suspended Enterprise
10. Judgement History
11. Related to Previous Participants
12. Suspicious Committee

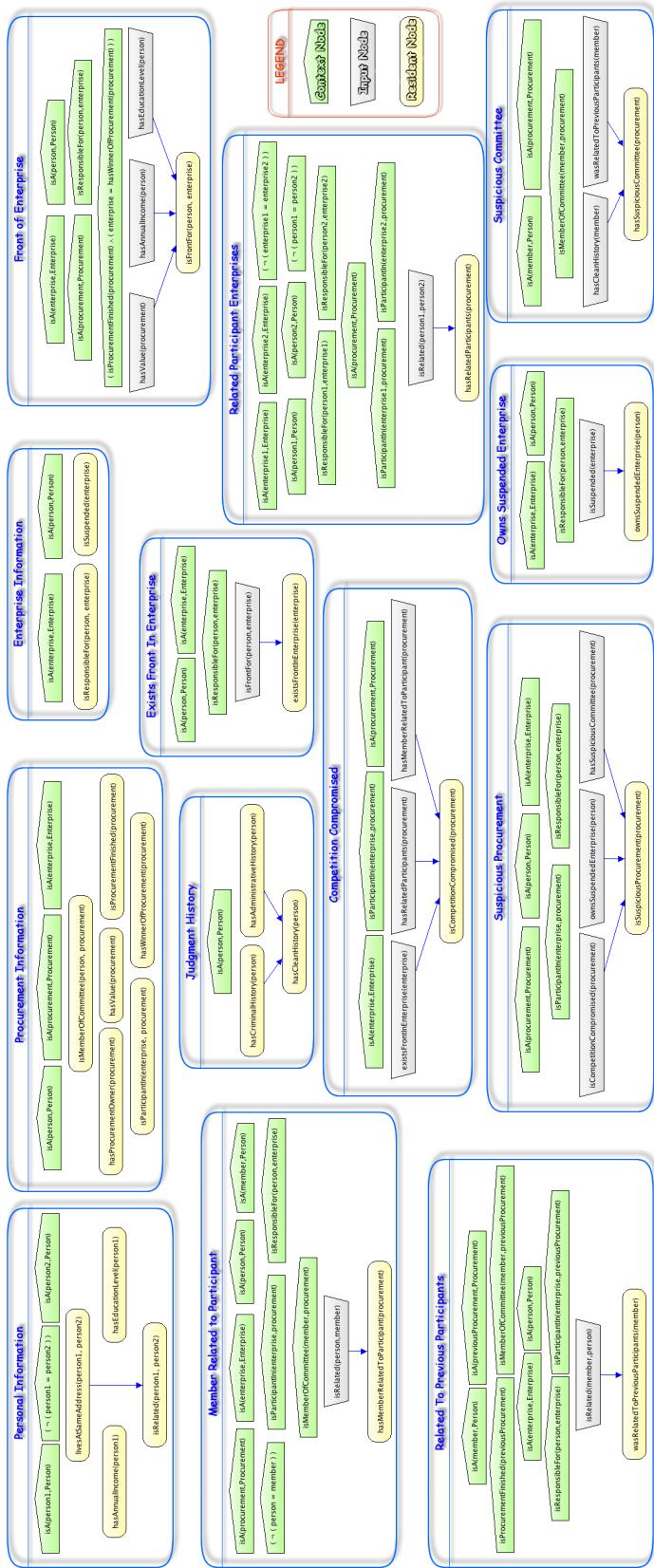


Figure 5.8: Probabilistic ontology for fraud detection and prevention in public procurements.

13. Suspicious Procurement

Table 5.3: Requirements Traceability Matrix for the MFrags of the fraud detection model.

ID	1	2	3	4	5	6	7	8	9
1	X	X	X		X	X		X	X
2	X	X	X	X	X	X	X	X	X
3	X	X	X	X	X	X		X	X
4			X			X			
5			X			X			
6					X	X			
7	X	X				X		X	X
8	X	X	X		X	X		X	X
9				X					
10							X		X
11								X	
12							X	X	
13	X	X	X	X	X	X	X	X	X

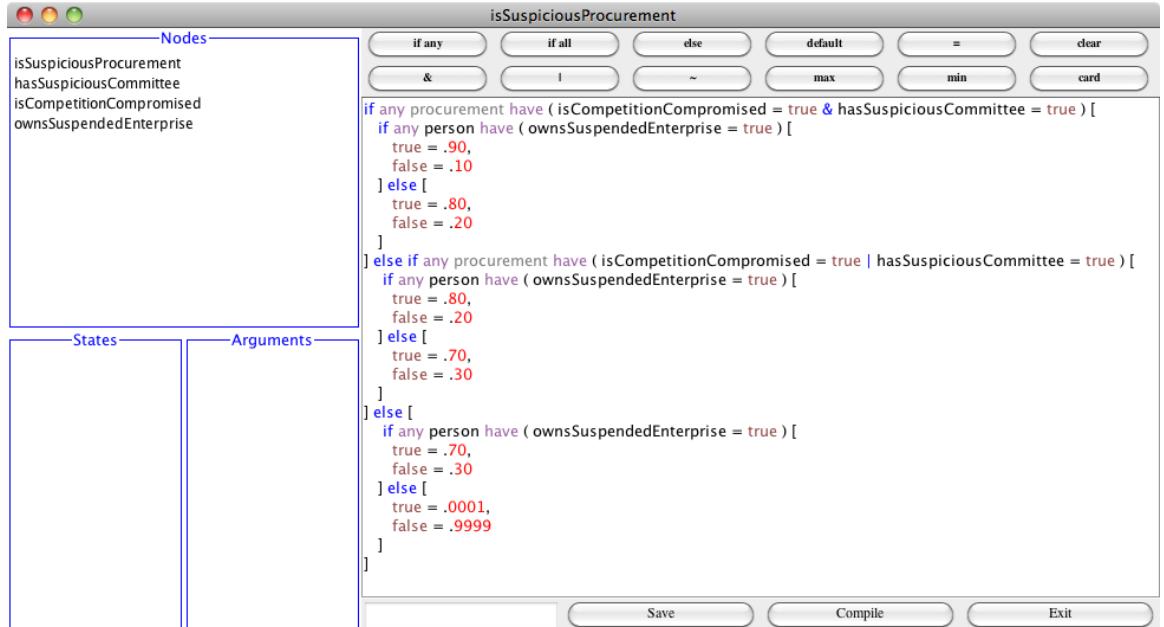
Table 5.3 presents the traceability between the MFrags defined in the Implementation stage and the rules defined in the Analysis & Design stage. This mapping, together with the mapping of the rules to the requirements presented in Table 5.2 provides the mapping that defines which requirements the MFrags are realizing.

Figure 5.8 presents an MTheory, in PR-OWL 1, that represents the final probabilistic ontology for the procurement fraud detection and prevention model. This MTheory is composed of nine MFrags. In each Mfrag, the resident RVs are shown as yellow rounded rectangles; the input RVs are shown as gray trapezoids; the context RVs are shown as green

pentagons. The two main goals described in our requirements are defined in the **Suspicious Procurement** and **Suspicious Committee** MFrags. A more sophisticated design to model whether to do further investigation or whether to change the committee would define a utility function and use expected utility to make the decision. Future versions of UnBBayes will support Multi-Entity Influence Diagrams [27].

The final step in constructing a probabilistic ontology in UnBBayes is to define the local probability distribution (LPD) for all resident nodes (in PR-OWL 2 the default distribution is defined only once on the RV itself). Figure 5.9 presents a LPD for the resident node **isSuspiciousProcurement(procurement)**, which is the main question we need to answer in order to achieve one of the main goals in our model. This distribution follows UnBBayes-MEBN expressive grammar for defining LPDs. For more information see [16, 19].

Appendix B Section B.1 presents the details and explanations of all MFrags and all resident nodes and their respective LPDs of the probabilistic ontology discussed in this Section.



The screenshot shows the UnBBayes software interface. At the top, there's a toolbar with buttons for 'if any', 'if all', 'else', 'default', '=', 'clear', '&', '|', '~', 'max', 'min', and 'card'. Below the toolbar is a text editor containing the LPD code for the node 'isSuspiciousProcurement'. The code uses MEBN-like syntax with conditionals and probability assignments. On the left side of the interface, there are three panels: 'Nodes' (containing 'isSuspiciousProcurement', 'hasSuspiciousCommittee', 'isCompetitionCompromised', and 'ownsSuspendedEnterprise'), 'States' (empty), and 'Arguments' (empty). At the bottom right are buttons for 'Save', 'Compile', and 'Exit'.

```

isSuspiciousProcurement
if any procurement have ( isCompetitionCompromised = true & hasSuspiciousCommittee = true ) [
  if any person have ( ownsSuspendedEnterprise = true ) [
    true = .90,
    false = .10
  ] else [
    true = .80,
    false = .20
  ]
] else if any procurement have ( isCompetitionCompromised = true | hasSuspiciousCommittee = true ) [
  if any person have ( ownsSuspendedEnterprise = true ) [
    true = .80,
    false = .20
  ] else [
    true = .70,
    false = .30
  ]
] else [
  if any person have ( ownsSuspendedEnterprise = true ) [
    true = .70,
    false = .30
  ] else [
    true = .0001,
    false = .9999
  ]
]

```

Figure 5.9: LPD for node **isSuspiciousProcurement(procurement)**.

5.1.4 Test

In most modeling methodologies, test plays an essential role. This is no different in the UMP-ST methodology. As Laskey and Mahoney [81] point out, test should not just be for showcase and to demonstrate that the model works. The Test discipline goal is to find flaws and areas for improvement in the model.

Before we start describing the activities in the Test discipline, it is important to understand the different types of evaluation that need to be done. The literature distinguishes two types of evaluation, verification and validation [6]. On the one hand, verification is concerned with delivering all the functionality promised to the customer. This usually involves reviewing requirements, documentation, design, and code. Verification is often done through inspections and by following checklists. On the other hand, validation is concerned with the correct behavior of the system. Validation is the actual testing of the system and it is done after verification.

A common slogan that summarizes the main difference between verification and validation is that verification tests whether the system was built right; validation tests whether we built the right system.

For instance, in the model we have been describing in this Section we would like to verify that all queries covered by the requirement are indeed being answered in less than a minute and that the posterior probability given as an answer to a given query is either exact or has an approximation with an error bound of .5% or less. These are non-functional requirements described during our Requirements stage in Subsection 5.1.1.

Although verification is an important and necessary evaluation, I will focus on describing how to validate our model. Laskey and Mahoney [81] present three types of validation: elicitation review, importance analysis, and case-based evaluation.

Elicitation review is related to reviewing the model documentation, analysing if all the requirements were addressed on the final model, making sure all the rules defined during the Analysis & Design stage were implemented, validating the semantics of the concepts described by the model, etc. This is an important step towards achieving consistency in

our model, especially if it was designed by more than one expert.

A good way to verify if all the requirements were addressed in the final implementation of the model is to look at the RTM matrices. By looking at the RTM matrix for the MFrags implemented in our model we can verify that all the rules defined during Analysis & Design were covered. Since the RTM matrix of the rules defined during Analysis & Design covered all the requirements, then we can infer that all the requirements were implemented in our model.

Importance analysis measures the strength of a link between nodes using some kind of sensitivity analysis method [75,96]. According to [81], “importance analysis for a given variable (called *focus* variable) measures the impact on the focus variable’s belief of obtaining evidence about each of a set of other variables (the *evidence* variables).”

In this section I will focus on case-based evaluation, which is defining different scenarios to test our model. One type of case-based evaluation is case-based unit testing. In case-based unit testing we want to test the behavior of part of the model, more specifically, verifying how the focus variable behaves with different set of evidence. In the case of PR-OWL, we can analyze the behavior of the random variables of interest given evidence per Mfrag. This Mfrag testing is important to capture local consistency of the model.

As an example of unit testing, I demonstrate how to define different scenarios to test the `JudgmentHistory_MFrag`. Essentially, we want to verify how the query `hasCleanHistory(person)` will behave in light of different set of evidence for a person’s criminal and administrative history.

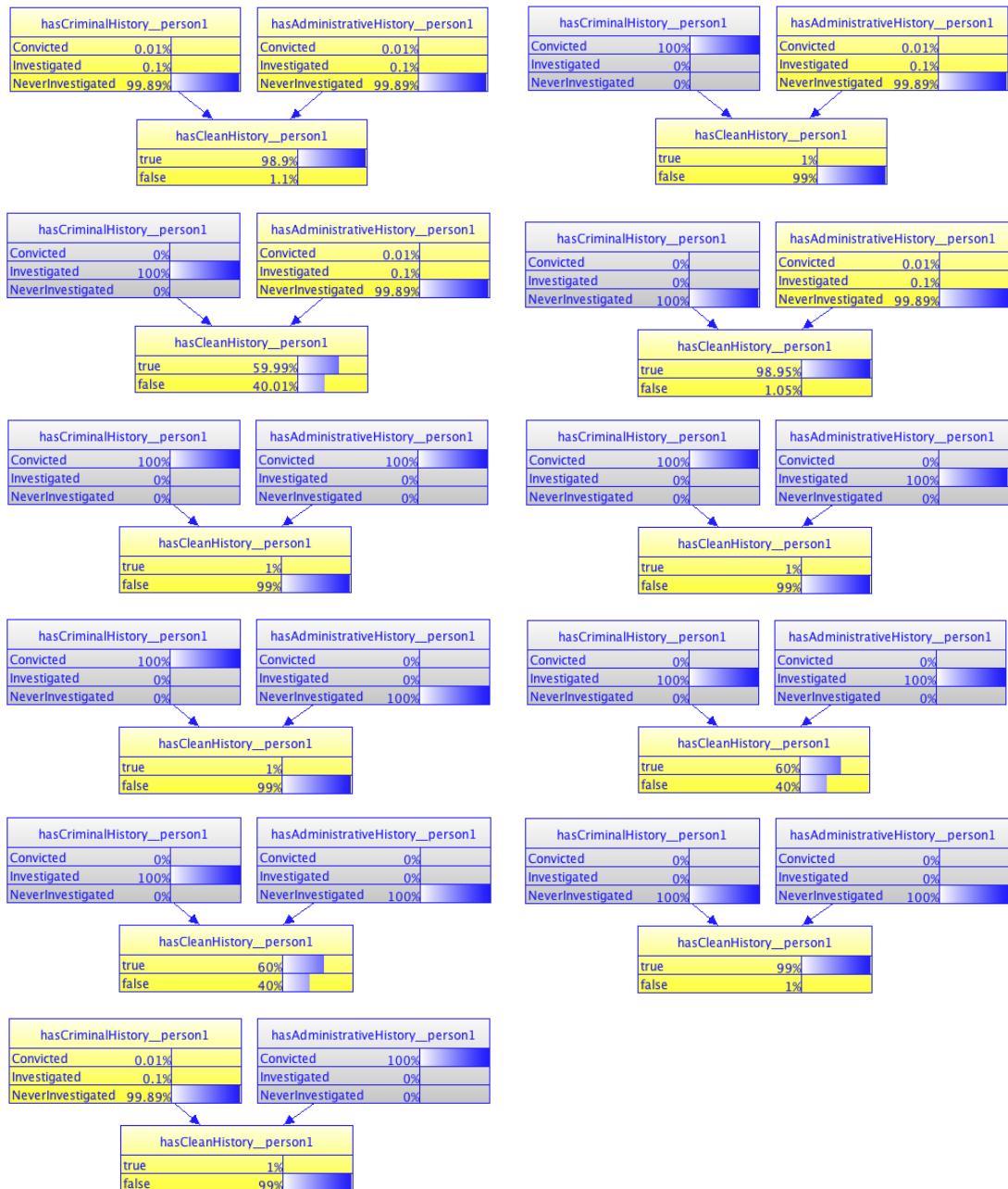


Figure 5.10: Results of unit testing for the JudgmentHistory_MFrag.

Notice that we do not show all possible combinations of the states for each node

in Figure 5.10, since their behavior is similar in the sense that stating that `hasCriminalHistory(person1) = Convicted` and `hasAdministrativeHistory(person1) = Investigated` is the same thing as stating that `hasCriminalHistory(person1) = Investigated` and `hasAdministrativeHistory(person1) = Convicted`, and so on. The important thing to do is to try to cover as much as possible and to analyze the results by verifying if the posterior probabilities behave as expected. In our case, the posterior probabilities are consistent with the expected result as defined by the expert. In this Mfrag the focus variable is the child, however, in other M frags the focus variable might be the parent and thus we would want to evaluate the behavior of a parent node given evidence on the children, which is the opposite of what was done here.

The other type of case-based evaluation is concerned with the behavior of the model as a whole. As such, I use it as an important type of integration testing. In the case of PR-OWL, we can define scenarios with evidence that are represented in different M frags. So, when we ask a query, the SSBN construction will instantiate different parts of the model, which helps us validate how the model works as a whole, and not just each part independently. This validation is important to capture global consistency of the model.

It is important to try out different scenarios in order to capture the nuances of the model. In fact, it is a good practice to design the scenarios in order to cover the range of requirements the model must satisfy [134, 124]. Although it is impossible to cover every scenario we might encounter, we should aim for good coverage, and especially look for important "edge cases". In order to illustrate this approach, let's define three different scenarios. The first one concerns a regular procurement with no evidence to support the hypothesis of a suspicious procurement or committee. The second one has conflicting evidence in the sense that some supports the hypothesis of having a suspicious procurement or committee but some does not. Finally, on the third scenario there is overwhelming evidence supporting the hypothesis of a suspicious procurement or committee. Nevertheless, a serious and more comprehensive evaluation of the model would have more than just three scenarios.

When defining a scenario, it is important to define the hypothesis being tested and what

is the expected result, besides providing the evidence which will be used. In this use case I was the subject matter expert, since I work for the Brazilian Office of the Comptroller General (CGU), which is the Government Agency responsible for supervising and auditing projects which involve federal money.

In the first scenario we have the following:

1. Hypothesis being tested

- (a) `isSuspiciousProcurement(procurement)`
- (b) `isSuspiciousCommittee(procurement)`

2. Expected result

- (a) Low probability that `isSuspiciousProcurement(procurement1) = true`
- (b) Low probability that `isSuspiciousCommittee(procurement1) = true`

3. Evidence

- (a) `hasAdministrativeHistory(member1) = NeverInvestigated`
- (b) `hasCriminalHistory(member2) = NeverInvestigated`
- (c) `hasProcurementOwner(procurement1) = agency1`
- (d) `isMemberOfCommittee(member1, procurement1) = true`
- (e) `isMemberOfCommittee(member2, procurement1) = true`
- (f) `isMemberOfCommittee(member3, procurement1) = true`
- (g) `isParticipantIn(enterprise1, procurement1) = true`
- (h) `isParticipantIn(enterprise2, procurement1) = true`
- (i) `isParticipantIn(enterprise3, procurement1) = true`
- (j) `isProcurementFinished(procurement1) = false`
- (k) `isResponsibleFor(person1, enterprise1) = true`

- (l) `isResponsibleFor(person2, enterprise2) = true`
- (m) `isResponsibleFor(person3, enterprise3) = true`

Figure 5.11 presents part of the SSBN network generated from scenario 1 and as expected the probability of both `isSuspiciousProcurement(procurement1) = true` and `isSuspiciousCommittee(procurement1) = true` are low, 2.35% and 2.33%, respectively.

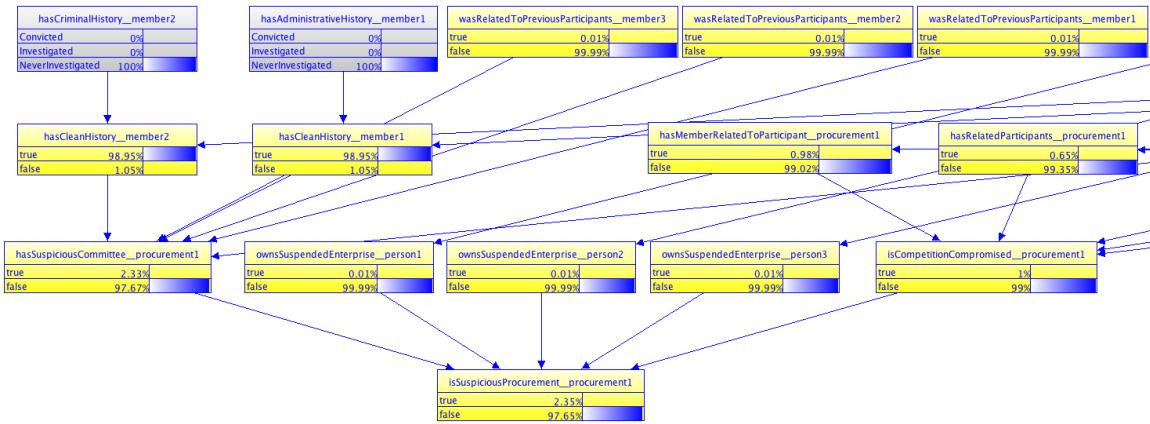


Figure 5.11: Part of the SSBN generated for the first scenario.

In the second scenario we have the following:

1. Hypothesis being tested
 - (a) `isSuspiciousProcurement(procurement)`
 - (b) `isSuspiciousCommittee(procurement)`
2. Expected result
 - (a) Probability that `isSuspiciousProcurement(procurement1) = true` between 10% and 50%
 - (b) Probability that `isSuspiciousCommittee(procurement1) = true` between 10% and 50%

3. Evidence (in italic we have the new evidence compared to scenario 1)

- (a) *hasAdministrativeHistory(member1) = Investigated*
- (b) *hasAdministrativeHistory(member1) = NeverInvestigated*
- (c) *hasCriminalHistory(member2) = NeverInvestigated*
- (d) *hasProcurementOwner(procurement1) = agency1*
- (e) *isMemberOfCommittee(member1, procurement1) = true*
- (f) *isMemberOfCommittee(member2, procurement1) = true*
- (g) *isMemberOfCommittee(member3, procurement1) = true*
- (h) *isParticipantIn(enterprise1, procurement1) = true*
- (i) *isParticipantIn(enterprise2, procurement1) = true*
- (j) *isParticipantIn(enterprise3, procurement1) = true*
- (k) *isProcurementFinished(procurement1) = false*
- (l) *isResponsibleFor(person1, enterprise1) = true*
- (m) *isResponsibleFor(person2, enterprise2) = true*
- (n) *isResponsibleFor(person3, enterprise3) = true*

Figure 5.12 presents part of the SSBN network generated from scenario 2 and as expected the probability of both *isSuspiciousProcurement(procurement1) = true* and *isSuspiciousCommittee(procurement1) = true* are 20.82% and 28.95%, respectively.

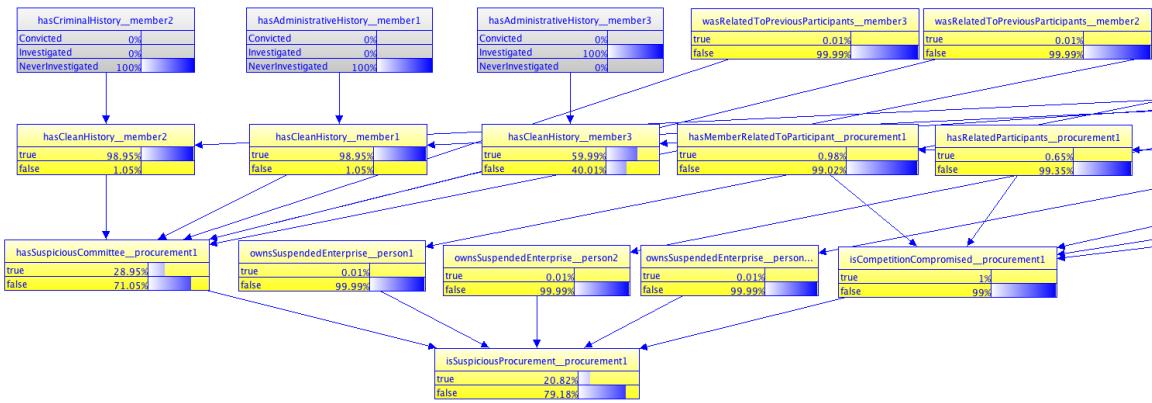


Figure 5.12: Part of the SSBN generated for the second scenario.

In the third scenario we have the following:

1. Hypothesis being tested
 - (a) `isSuspiciousProcurement(procurement)`
 - (b) `isSuspiciousCommittee(procurement)`
2. Expected result
 - (a) Probability that `isSuspiciousProcurement(procurement1) = true` greater than 50%
 - (b) Probability that `isSuspiciousCommittee(procurement1) = true` between 10% and 50%
3. Evidence (in italic we have the new evidence compared to scenario 2)
 - (a) `livesAtSameAddress(person1, person3)`
 - (b) `livesAtSameAddress(person2, member3)`
 - (c) `hasAdministrativeHistory(member1) = Investigated`
 - (d) `hasAdministrativeHistory(member1) = NeverInvestigated`

- (e) `hasCriminalHistory(member2) = NeverInvestigated`
- (f) `hasProcurementOwner(procurement1) = agency1`
- (g) `isMemberOfCommittee(member1, procurement1) = true`
- (h) `isMemberOfCommittee(member2, procurement1) = true`
- (i) `isMemberOfCommittee(member3, procurement1) = true`
- (j) `isParticipantIn(enterprise1, procurement1) = true`
- (k) `isParticipantIn(enterprise2, procurement1) = true`
- (l) `isParticipantIn(enterprise3, procurement1) = true`
- (m) `isProcurementFinished(procurement1) = false`
- (n) `isResponsibleFor(person1, enterprise1) = true`
- (o) `isResponsibleFor(person2, enterprise2) = true`
- (p) `isResponsibleFor(person3, enterprise3) = true`

Figure 5.13 presents part of the SSBN network generated from scenario 3 and as expected the probability of both `isSuspiciousProcurement(procurement1) = true` and `isSuspiciousCommittee(procurement1) = true` are 60.08% and 28.95%, respectively.

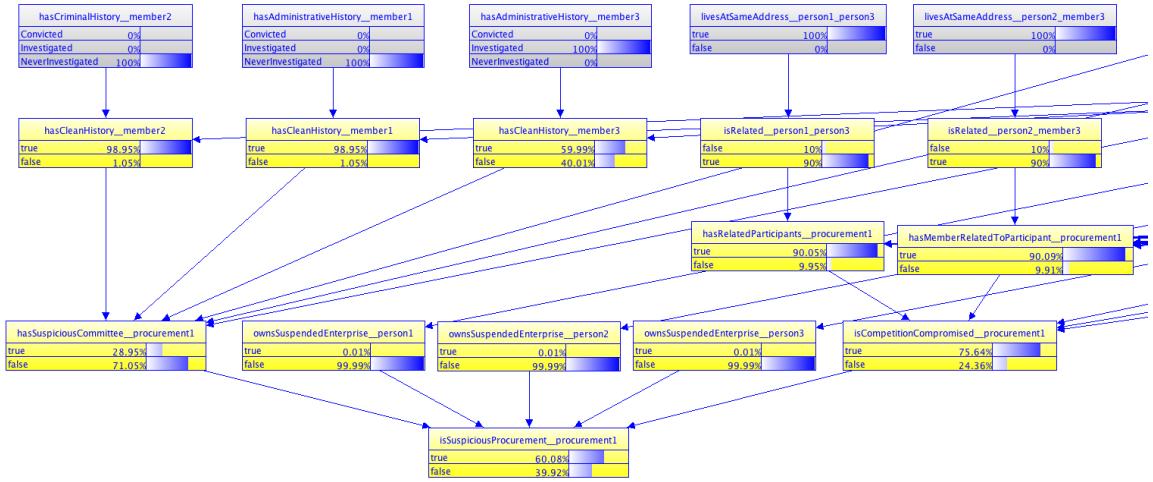


Figure 5.13: Part of the SSBN generated for the third scenario.

5.2 Probabilistic Ontology for Maritime Domain Awareness

Maritime Domain Awareness (MDA) involves the ability to automatically integrate information from multiple sources in a complex and evolving scenario to produce a dynamic, comprehensive, and accurate picture of the naval operations environment. The emphasis on net-centric operations and the shift to asymmetric warfare have added an additional level of complexity and technical challenge to automated information integration and predictive situation assessment. A probabilistic ontology (PO) is a promising tool to address this challenge. The PO for Maritime Domain Awareness (MDA) described in this Section was presented in [17, 18] and is part of the PROGNOS project [32, 33].

PROGNOS (PRobabilistic OntoloGies for Net-centric Operation Systems) is a naval predictive situational awareness system devised to work within the context of U.S. Navy's FORCENet. The system uses the UnBBayes-MEBN framework, which implements a MEBN reasoner capable of saving MTheories in PR-OWL format.

The focus of this Section is to highlight the key role iterations play in incrementally expanding the model during its lifecycle. In this Section I will not present as much detail

in each discipline as I did in Section 5.1. Instead I will highlight how we can leverage the UMP-ST process and PR-OWL’s modularity in order to minimize change in the existing model as we add new requirements in new iterations.

The PROGNOS MDA PO was created using the Uncertainty Model for Semantic Technologies (UMP-ST) and the Probabilistic Ontology Modeling Cycle (POMC) with the support of the stakeholders (MEBN and PR-OWL experts and subject matter experts, who are retired officers from US Navy and US Coast Guard, Richard Haberlin and Michael Lehocky, respectively). The probabilistic ontology developed so far has passed through three iterations. The first iteration consists of a simple model to identify whether a ship is of interest. The second iteration expanded the model to provide clarification of the reasons behind declaring a ship of interest. The third iteration focused on detecting an individual crew member’s terrorist affiliation given his close relations, group associations, communications, and background influences.

5.2.1 First Iteration

Requirements

The original model consists of the following set of goal/query/evidence:

1. Identify whether a ship is of interest, *i.e.*, it seems to be suspicious in any way.
 - (a) Does the ship have a terrorist crew member?
 - i. Verify if a crew member is related to any terrorist;
 - ii. Verify if a crew member is associated with any terrorist organization.
 - (b) Is the ship using an unusual route?
 - i. Verify if there is a report that the ship is using an unusual route;
 - ii. Verify if there is a report that the ship is meeting some other ship for no apparent reason.
 - (c) Does the ship seem to exhibit evasive behavior?

- i. Verify if an electronic countermeasure (ECM) was identified by a navy ship;
- ii. Verify if the ship has a responsive radio and automatic identification system (AIS).

Analysis & Design

Once we have defined our goals and described how to achieve them, it is time to start modeling the entities, their attributes, relationships, and rules to make that happen. This is the purpose of the Analysis & Design discipline.

Figure 5.14 depicts a simplified design of our domain requirements. A **Ship** is a ship of interest, **isOfInterest**, if it represents some kind of threat. A **Ship** has a crew, which is represented by **hasCrewmember** and the inverse relation **isCrewmemberOf**. It is assumed that a ship represents some kind of threat if and only if one of its crew members is a **Terrorist** (subclass of **Person**).

The social network information available determines that a **Person** might be related to another **Person** by **isRelatedTo**. Moreover, a **Person** might be a member of an **Organization**, represented by the **isMemberOf** and the inverse **hasMember** relations. An **Organization** might be a **TerroristOrganization** (subclass of **Organization**). It is also assumed that a **Person** related to a **Terrorist** is more likely to be a **Terrorist** and an **Organization** that has a **Terrorist** member is more likely to be a **TerroristOrganization**.

This model is simplified in the sense that it represents a screenshot in time of the domain. In other words, there is only one possible crew for a given **Ship** and a **Person** can only be a crew member of a unique **Ship**. Following the same rationale, a **Ship** can only have one possible **Position**, represented by **hasPosition**.

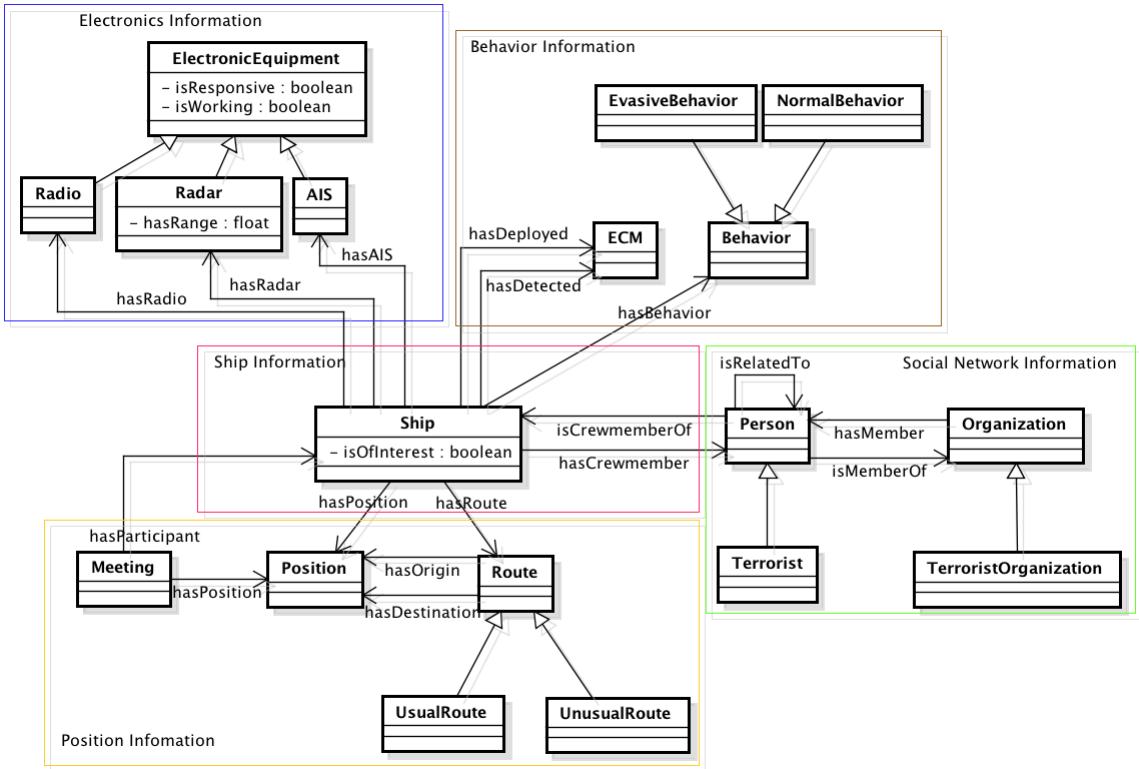


Figure 5.14: Entities, their attributes, and relations for the MDA model after the first iteration.

The **Position** of a **Ship** is usually consistent with its **Route**. A **Route** has a specific origin and destination **Position**, represented by `hasOrigin` and `hasDestination`, respectively. If the **Ship** is following the usual route from its origin to its destination, then its **Route** is said to be a **UsualRoute**, otherwise, if the **Ship** is going to places that are not consistent with the expected route (safest/shortest distance from origin to destination), then its **Route** is said to be an **UnusualRoute**. Furthermore, usually ships try to avoid getting too close to each other, therefore, if two or more ships get too close together, it is said that they are **Meeting** in a certain **Position**, represented by `hasPosition`. The ships participating in this **Meeting** are represented by `hasParticipant`, which maps a **Meeting** to two or more ships (**Ship**). If two or more ships are meeting, then it is more likely that they are doing some

illicit transaction on the ocean, therefore, they will probably meet at an unusual **Position**, which means that they are on an **UnusualRoute**. One example illustrating this idea is that a ship carrying Weapons of Mass Destruction (WMD) might want to pass its dangerous cargo to one or more smaller ships in order to increase the chances of infiltrating the coast with the WMD.

As for the electronic equipment described in this model, **ElectronicEquipment**, a **Ship** can have an Automatic Identification System (AIS), represented by **hasAIS**, which is used for identifying and locating vessels by electronically exchanging data with other nearby ships and Vessel Traffic Services (VTS) stations. Moreover, a **Ship** usually has at least one **Radar**, represented by **hasRadar**, with a specific range, defined by **hasRange**. The range is defined in this model by a **float** number, however, in a more realistic and detailed model this should be a measure of distance, *i.e.*, a class by itself with value and unit of measure. **AIS** and **Radar** are subclasses of **EletronicEquipment** and as such, they can be responsive, represented by **isResponsive**, which entails that they are working, represented by **isWorking**, and turned on.

A **Ship** might have different behaviors (**Behavior**). A **Ship** might deploy an Electronic Countermeasure (ECM), represented by **hasDeployed**. Besides that, a different **Ship** might detect an ECM, represented by **hasDetected**, although it does not necessarily know which **Ship** deployed it. To be able to detect an ECM, the ship that deployed the ECM has to be in the **Radar** range of the **Ship** that detects it. An ECM is a subsection of electronic warfare, which includes any sort of electrical or electronic device designed to trick or deceive radar, sonar, or other detection systems. It may be used both offensively and defensively in any method to deny targeting information to an enemy. A **Ship** that has deployed an ECM is said to have exhibited an **EvasiveBehavior**. Furthermore, if an **ElectronicEquipment** is working but is not responsive, then the **Ship** is also said to have exhibited an **EvasiveBehavior**. In all other cases, the **Ship** is said to have **NormalBehavior**. As shown, **EvasiveBehavior** and **NormalBehavior** are subclasses of **Behavior**.

Besides the cardinality and uniqueness rules defined in the explanation above about the

entities depicted in Figure 5.14, the probabilistic rules for our model include:

1. A ship is of interest if and only if it has a terrorist crew member;
2. If a crew member is related to a terrorist, then it is more likely that he is also a terrorist;
3. If a crew member is a member of a terrorist organization, then it is more likely that he is a terrorist;
4. If an organization has a terrorist member, it is more likely that it is a terrorist organization;
5. A ship of interest is more likely to have an unusual route;
6. A ship of interest is more likely to meet other ships for trading illicit cargo;
7. A ship that meets other ships to trade illicit cargo is more likely to have an unusual route;
8. A ship of interest is more likely to have an evasive behavior;
9. A ship with evasive behavior is more likely to have non responsive electronic equipment;
10. A ship with evasive behavior is more likely to deploy an ECM;
11. A ship might have non responsive electronic equipment due to working problems;
12. A ship that is within radar range of a ship that deployed an ECM might be able to detect the ECM, but not who deployed it.

Implementation

Once we have finished our Analysis & Design, it is time to start implementing our model in a specific language. In this project we implement our model in PR-OWL using UnBBayes.

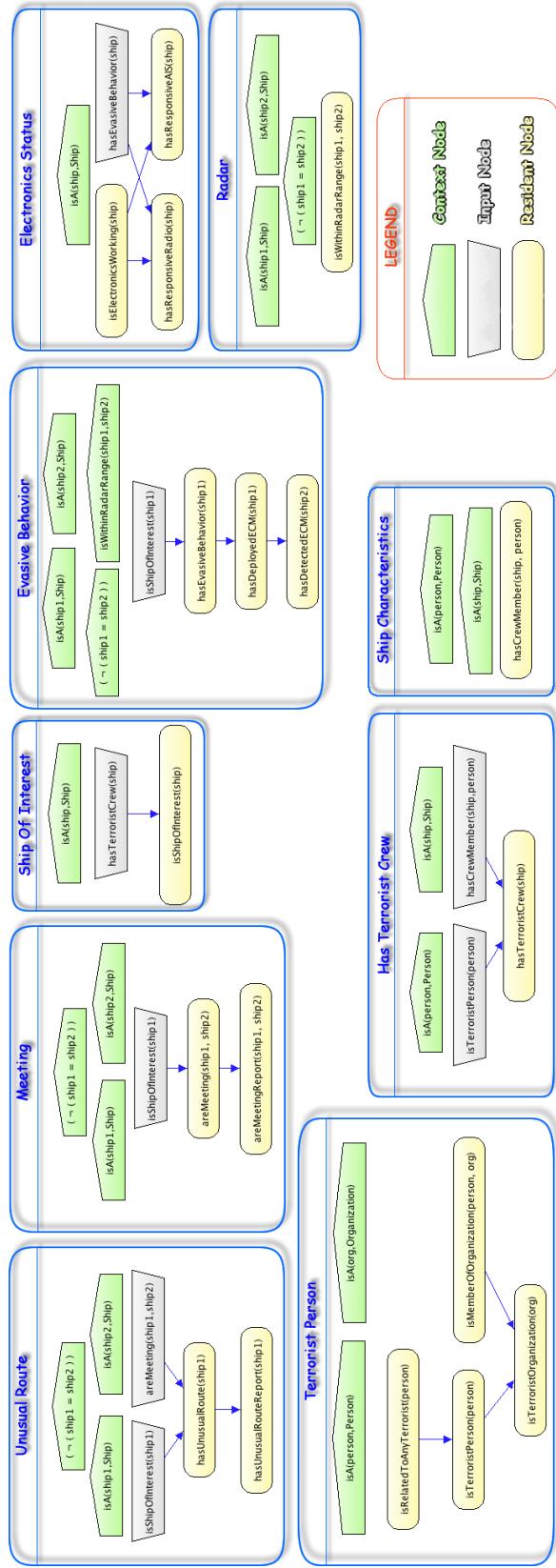


Figure 5.15: MTheory created in first iteration.

The final result of this initial iteration is the PO depicted in Figure 5.15. There, the hypotheses related to the identification of a terrorist crew member are presented in the `Has Terrorist Crew`, `Terrorist Person`, and `Ship Characteristics` MFrags. The hypotheses related to the identification of unusual routes are presented on the `Unusual Route` and `Meeting` MFrags. Finally, the hypotheses related to identification of evasive behavior are shown in the `Evasive Behavior`, `Electronics Status`, and `Radar` MFrags.

Appendix B Subsection B.2.1 presents the details and explanations of all MFrags and all resident nodes and their respective LPDs of the probabilistic ontology discussed in this Subsection.

Test

Although I have described many different types of evaluation and tests we can perform in our model in Subsection 5.1.4, this iteration will focus on performing integration test based on case-based evaluation.

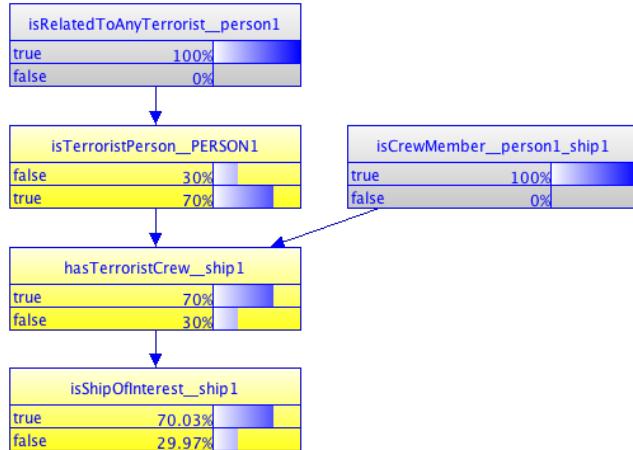


Figure 5.16: SSBN generated for scenario 1.

I will illustrate 5 different scenarios by increasing not only the complexity of the generated model, but also the probability that `ship1` is of interest. These increases are due to

new evidence that is available in every new scenario, which supports the hypothesis that **ship1** is of interest.

In scenario 1, the only information available is that **person1** is a crew member of **ship1** and that **person1** is related to at least one terrorist. Figure 5.16 shows that there is a 70.03% probability of **ship1** being of interest, which is consistent with the fact that one of its crew members might be a terrorist.

In scenario 2, besides having the information available from scenario 1, it is also known that **ship1** met **ship2**. Figure 5.17 shows the probability of **ship1** being of interest has increased to 89.41%, which is consistent with the new supporting evidence that **ship1** met **ship2**.

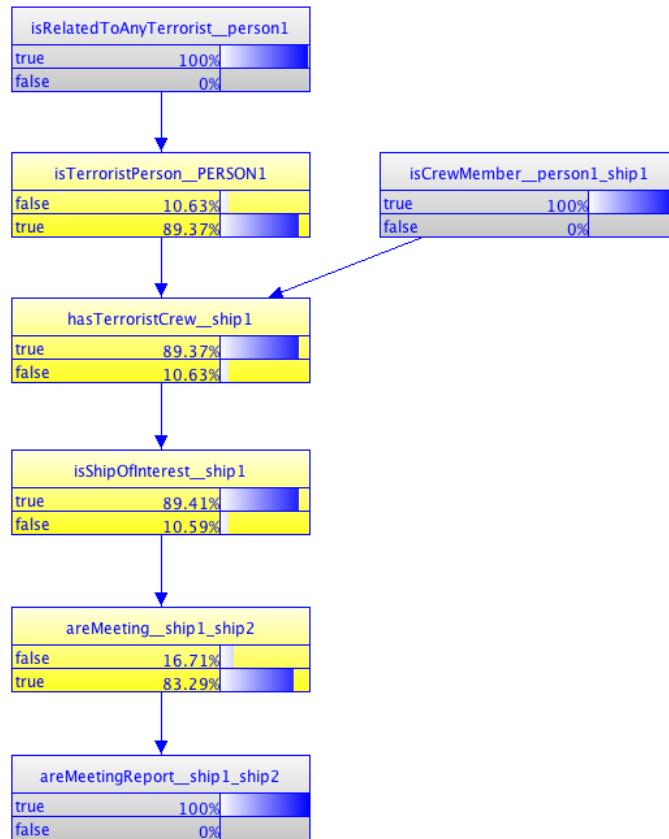


Figure 5.17: SSBN generated for scenario 2.

In scenario 3, besides having the information available from scenario 2, it is also known that **ship1** has an unusual route. Figure 5.18 shows the probability of **ship1** being of interest has increased to 97.19%, which is consistent with the new supporting evidence that **ship1** is not going to its destination using a normal route.

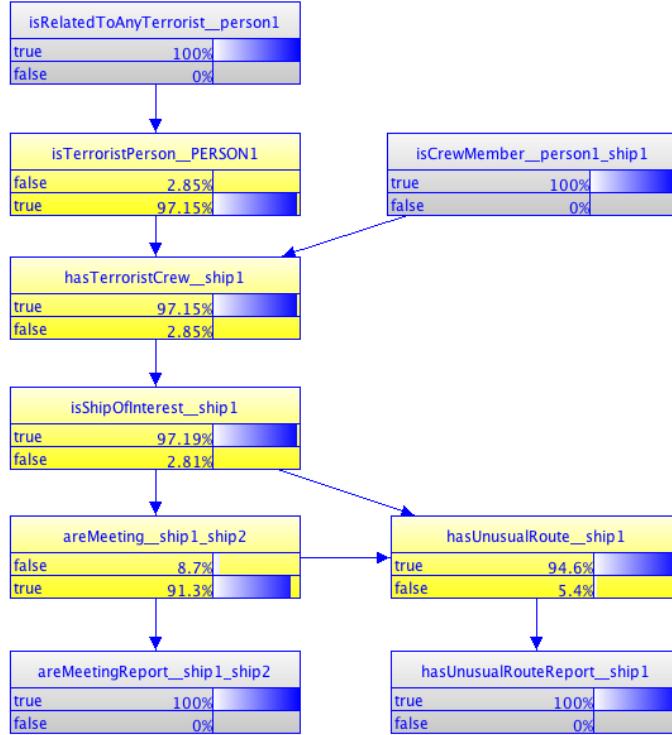


Figure 5.18: SSBN generated for scenario 3.

In scenario 4, besides having the information available from scenario 3, it is also known that **navyShip** has detected an ECM. Figure 5.19 shows the probability of **ship1** being of interest has increased to 99.97%, which is consistent with the new supporting evidence that **ship1** is probably the ship that deployed the ECM. It is important to notice that there are only two ships that could deploy the ECM in this scenario, which are the ships within range of **navyShips** radar (**ship1** and **ship2**). From the other evidence that supports the fact that **ship1** is most likely a ship of interest, it becomes more likely that **ship1** is the

one that deployed the ECM. That is why the probability that **ship2** having deployed the ECM is so low (due to explaining away).

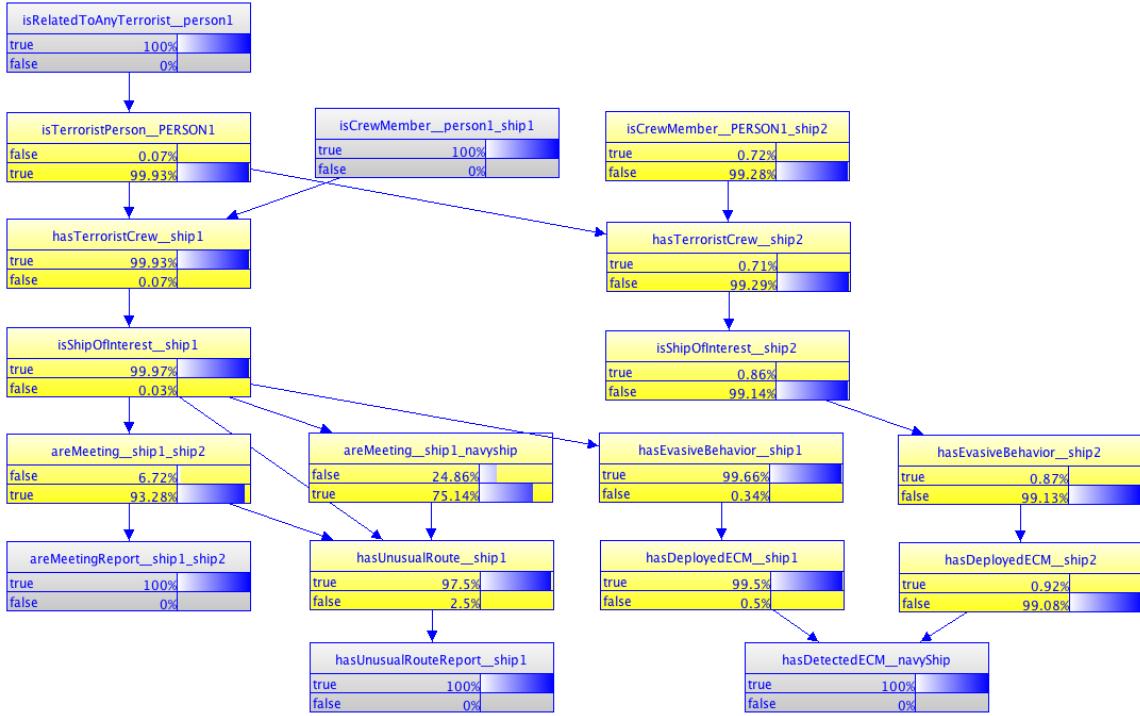


Figure 5.19: SSBN generated for scenario 4.

In scenario 5, besides having the information available from scenario 4, it is also known that **ship1** does not have a responsive radio nor a responsive AIS. Figure 5.20 shows that the probability of **ship1** being of interest is 100.00%.

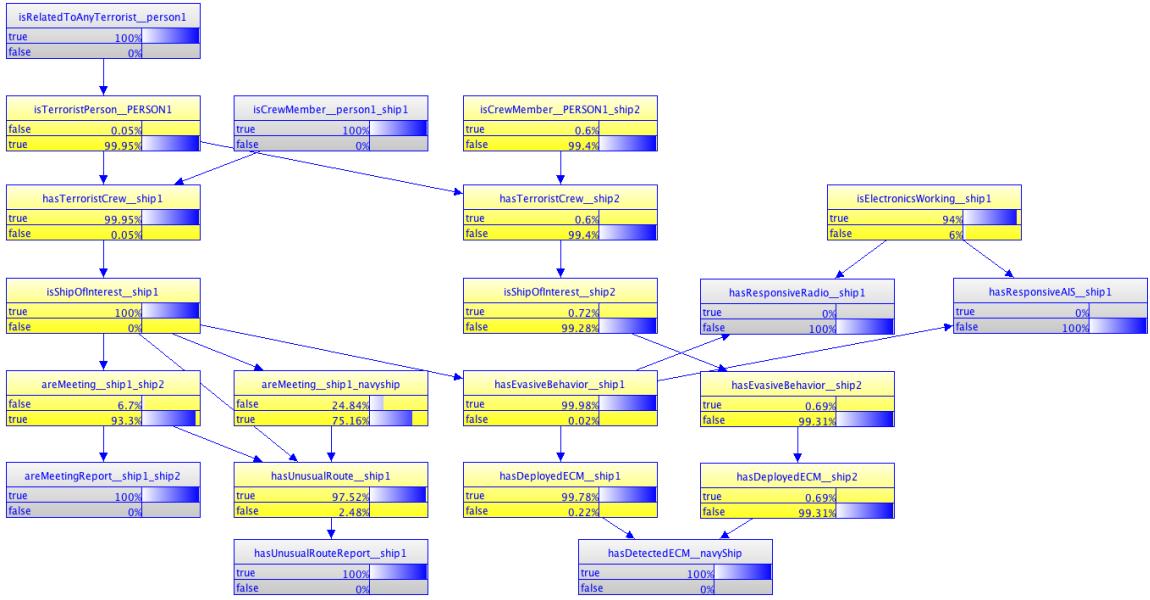
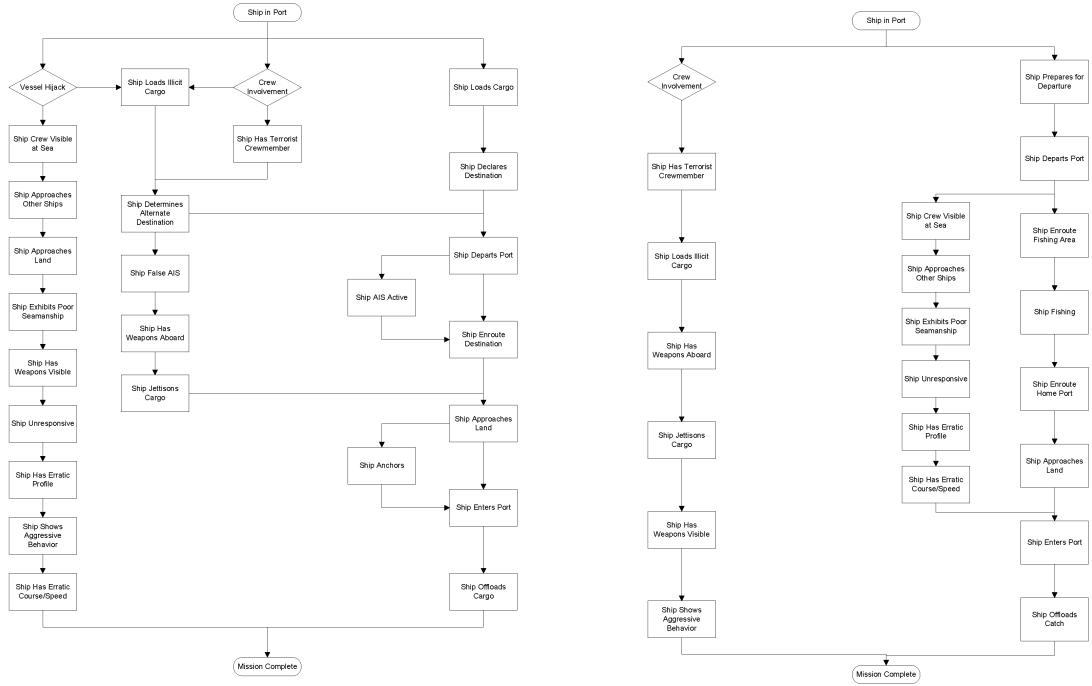


Figure 5.20: SSBN generated for scenario 5.

5.2.2 Second Iteration

Once the initial model was built and tested, the second iteration shifted focus to understanding the reasons for classifying a ship's behavior as suspicious. The approach was to define possible terrorist plans that might result in specific behaviors. At this stage, two terrorist plans were taken into consideration: exchange illicit cargo (*e.g.*, explosives) and bomb a port using a suicide ship. Another distinction from the original model is that the behavior depends not only on the plan being executed, but also on the type of the ship. In addition, there are now two reasons why a ship might be executing a terrorist plan: it either has a terrorist crew member (the only option in the original model) or the ship was hijacked.



(a) Merchant ship with exchange illicit cargo plan on the left, and normal behavior on the right.
(b) Fishing ship with bomb a port plan on the left, and normal behavior on the right.

Figure 5.21: Normal and suspicious behavior of merchant and fishing ships.

Figure 5.21 provides an activity diagram with the expected behaviors of ships involved in illicit activities on the left, and what would be the normal behavior from ships with no terrorist plan on the right.

Requirements

With the new task of identifying the terrorist plans associated to a suspicious ship (*i.e.*, exchanging illicit cargo, bombing a port, or no terrorist plan), the second iteration's set of goal/query/evidence was also expanded:

Identify whether a ship is a ship of interest, *i.e.*, if the ship has some terrorist plan associated with it.

1. Is the ship being used to exchange illicit cargo?

- (a) Was the ship hijacked?
 - (b) *Does the ship have a terrorist crew member?*
 - i. *Verify if a crew member is related to any terrorist;*
 - ii. *Verify if a crew member is associated with any terrorist organization.*
 - (c) *Is the ship using an unusual route?*
 - i. *Verify if there is a report that the ship is using an unusual route;*
 - ii. *Verify if there is a report that the ship is meeting some other ship for no apparent reason.*
 - iii. Verify if the ship had a normal change in destination (*e.g.*, to sell the fish, which was just caught.)
 - (d) *Does the ship seem to exhibit evasive behavior?*
 - i. *Verify if an electronic countermeasure (ECM) was identified by a navy ship;*
 - ii. *Verify if the ship has a responsive radio and automatic identification system (AIS).*
 - (e) Does the ship seem to exhibit erratic behavior?
 - i. Verify if the crew of the ship is visible.
 - (f) Does the ship seem to exhibit aggressive behavior?
 - i. Verify if the ship has weapons visible;
 - ii. Verify if the ship is jettisoning cargo.
2. Is the ship being used as a suicide ship to bomb a port?
- (a) Was the ship hijacked?
 - (b) *Does the ship have a terrorist crew member?**
 - (c) *Is the ship using an unusual route?**
 - (d) Does the ship seem to exhibit aggressive behavior?*

Requirements inherited from the first iteration are in italic. Items crossed out refer to evidence considered by the SMEs, but that pertain only to war ships. Since these are not included in the scenarios they were excluded from the model. Queries marked with '*' are also used for another subgoal. For instance, an unusual route is expected both from ships with plan to bomb a port and from ships planning to exchange illicit cargo. The associated evidence is shown only for the first subgoal using the query.

Analysis & Design

As the original requirements were expanded, the UML model was also expanded to identify new concepts needed for achieving the new goals. Figure 5.22 displays the resulting model, with some classes added (*e.g.*, Plan, TerroristPlan, TypeOfShip, etc) and others removed (*e.g.*, ECM). Major changes are the new types of behavior (AggressiveBehavior and ErraticBehavior), the classification of ships (TypeOfShip and its subclasses), and planning information (Plan, TerroristPlan, and its subclasses). In addition, class Ship was expanded to allow for situational awareness of its behavior and to predict future actions based on it.

The next step is to define rules associated with the new requirements. The probabilistic rules below complement the cardinality and uniqueness rules in Figure 5.22 (same typing convention for rules inherited or not used in the model apply).

1. *A ship is of interest if and only if it has a terrorist crew member plan;*
2. A ship has a terrorist plan if and only if it has terrorist crew member or if it was hijacked;
3. *If a crew member is related to a terrorist, then it is more likely that he is also a terrorist;*
4. *If a crew member is a member of a terrorist organization, then it is more likely that he is a terrorist;*

5. *If an organization has a terrorist member, it is more likely that it is a terrorist organization;*
6. *A ship of interest is more likely to have an unusual route, independent of its intention;*
7. *A ship of interest, with plans of exchanging illicit cargo, is more likely to meet other ships;*
8. *A ship that meets other ships to trade illicit cargo is more likely to have an unusual route;*
9. A fishing ship is more likely to have a normal change in its destination (*e.g.*, to sell the fish caught) than merchant ships;
10. A normal change in destination will probably change the usual route of the ship;
11. *A ship of interest, with plans of exchanging illicit cargo, is more likely to have an evasive behavior;*
12. A ship with evasive behavior is more likely to have non responsive electronic equipment;
13. A ship might have non responsive electronic equipment due to maintenance problems;
14. ~~A ship with evasive behavior is more likely to deploy an ECM;~~
15. ~~A ship that is within radar range of a ship that deployed an ECM might be able to detect the ECM, but not who deployed it;~~
16. A ship of interest, with plans of exchanging illicit cargo, is more likely to have an erratic behavior;
17. A ship with normal behavior usually does not have the crew visible on the deck;
18. A ship with erratic behavior usually has the crew visible on the deck;

19. If the ship has some equipment failure, it is more likely to see the crew on the deck in order to fix the problem;
20. A ship of interest, independent of its intention, is more likely to have an aggressive behavior;
21. A ship with aggressive behavior is more likely to have weapons visible and to jettison cargo;
22. A ship with normal behavior is not likely to have weapons visible nor to jettison cargo.

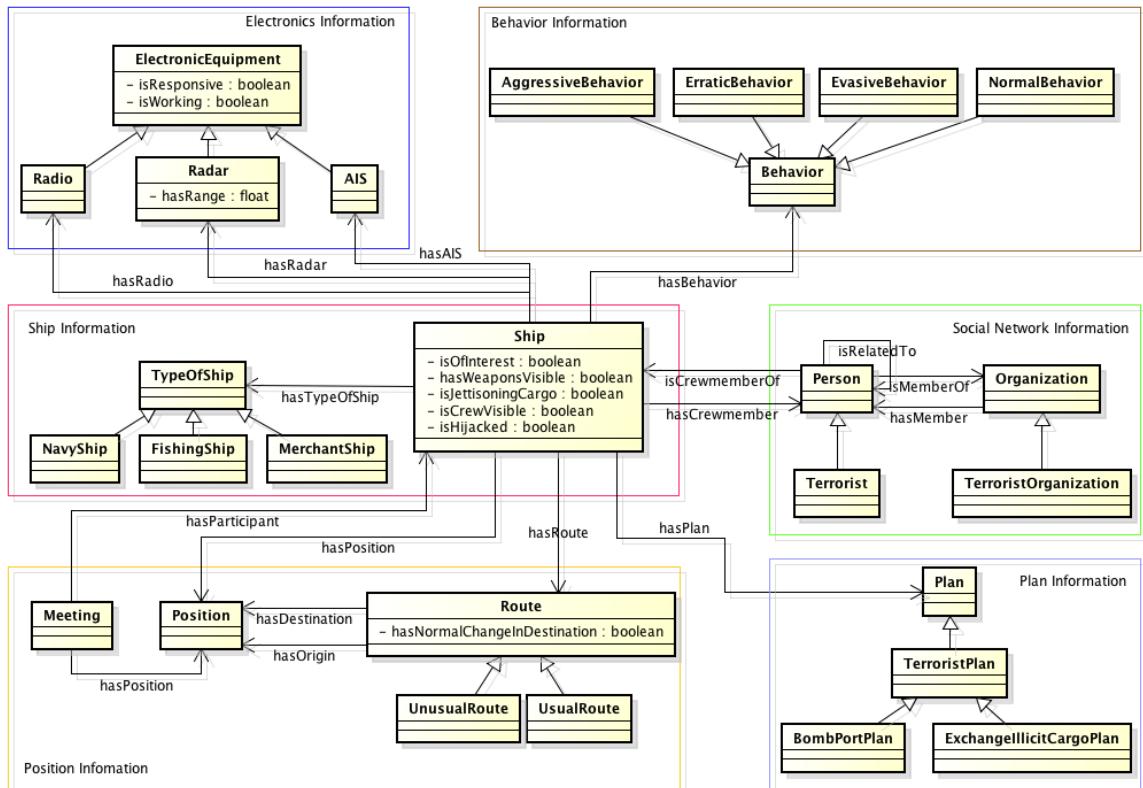


Figure 5.22: Entities, their attributes, and relations for the MDA model after the second iteration.

Implementation

Once the Analysis and Design stage is finished, implementation in a specific language (PR-OWL in this case) begins. The initial step is to map entities, attributes, and relations to PR-OWL. There is no need to map all entities in the model to entities in PR-OWL

1. In fact, the MDA model contains many simplifications. One is to define the random variable `hasTypeOfShip` mapping to values `Fishing` or `Merchant`, instead of creating them as subclasses. This can be done by creating a class in OWL using `oneOf` to specify the individuals that represent the class `ShipType`. Also, the original assumption of every entity being uniquely identified by its name still holds. The entities implemented in the MDA PO were `Person`, `Organization`, and `Ship`. All other entities were simplified in a similar manner as `ShipType`. For details on defining entities in UnBBayes see [20].

As explained in Subsection 5.1.3, in PR-OWL 2, it is not necessary to map these entities. In fact, the entities are defined as classes in a regular ontology using OWL. Then PR-OWL 2 simply makes use of them. As previously explained, our focus in this Section is to show how the model evolves when using the UMP-ST process, not on describing details on how to create a deterministic ontology.

After defining entities, the uncertain characteristics are identified. Uncertainty is represented in MEBN as random variables (RVs). On the one hand, to define a RV in PR-OWL 1 using UnBBayes, we first define its home Mfrag. Grouping the RVs into MFrags is done by examining the grouping created during Analysis & Design. On the other hand, in PR-OWL 2 RVs are independent of the MFrags containing them and are defined globally by defining their arguments, mapping to OWL, and default distributions.

Typically, a RV represents an attribute or a relation in the designed model. For instance, the RV `isHijacked(Ship)` maps to the attribute `isHijacked` of the class `Ship` and the RV `hasCrewMember(Ship, Person)` maps to the relation `hasCrewMember` (refer to Figure 5.22). As a predicate relation, `hasCrewMember` relates a `Ship` to one `Person` or more, the same way class `Ship` might have one `Person` or more as its crew members. Hence, the possible values (or states) of this RV are `True` or `False`. Subclasses were avoided by using Boolean

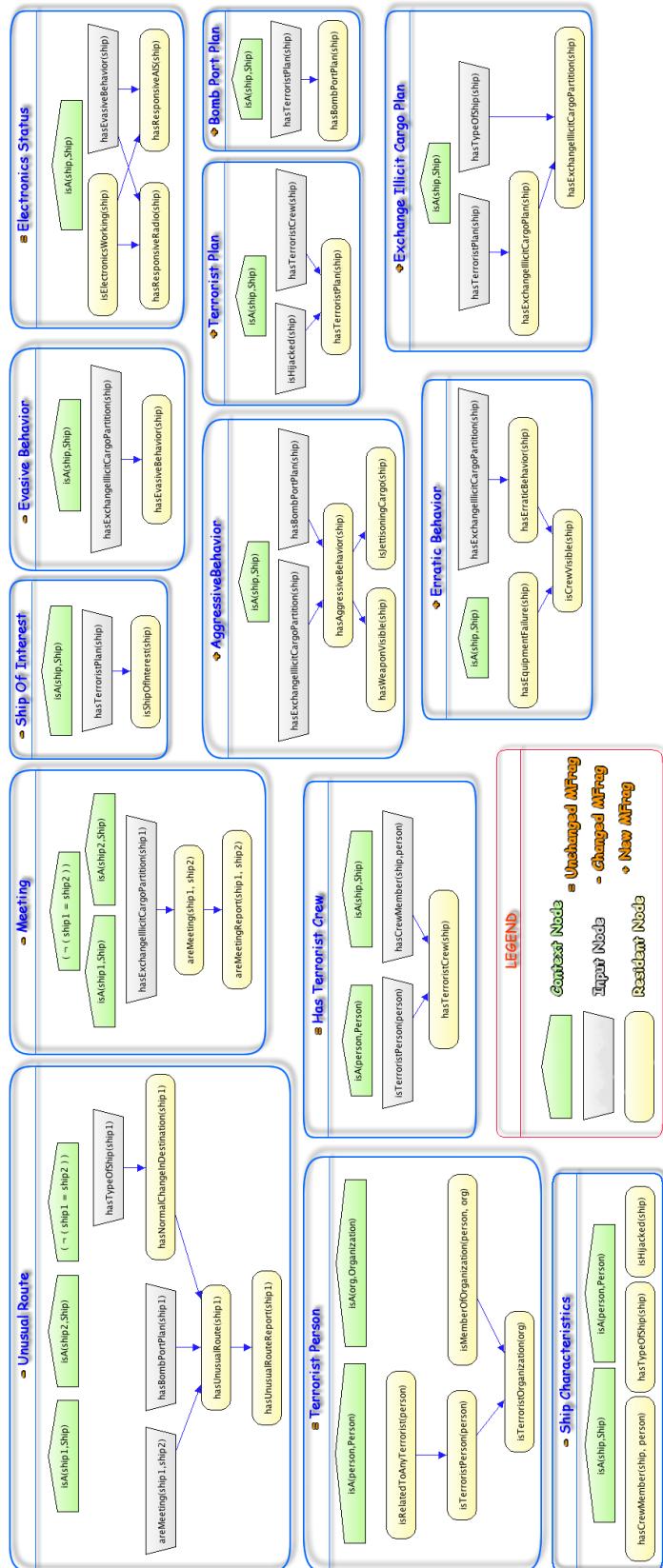


Figure 5.23: MTheory created in second iteration.

RV like `isTerrorist(Person)`, which represents the subclass `Terrorist`.

Once all resident RVs are created, their relations are defined by analyzing dependencies. This is achieved by looking at the rules defined in the model. For instance, the first rule indicates a dependence between `hasTerroristPlan(Ship)` and `isShipOfInterest(Ship)`. The structure of the relations added to the MDA PO can be seen in Figure 5.23.

After defining the relations, the local probability distributions are inserted for each resident node. For conciseness, these are not presented here but they must be consistent with the probabilistic rules defined in the Analysis & Design stage.

Appendix B Subsection B.2.2 presents the details and explanations of all MFrags and all resident nodes and their respective LPDs of the probabilistic ontology discussed in this Subsection.

Test

Although I have described many different types of evaluation and tests we can perform in our model in Subsection 5.1.4, this iteration will focus on performing integration test based on case-based evaluation, as was the case in the first iteration.

As explained in Subsection 5.1.4 it is important to try out different scenarios in order to capture the nuances of the model. In a serious test of the model, we would have to model a lot scenarios in order to cover at least the most important aspects of our requirements. However, I define only three qualitatively different scenarios in order to illustrate the mechanics of defining and testing a scenario. The first one has a regular ship with no evidence that supports the hypothesis of having a terrorist plan. The second one has conflicting evidence in the sense that some supports the hypothesis of having a terrorist plan but some does not. Finally, on the third scenario there is overwhelming evidence that supports the hypothesis of having a terrorist plan.

When defining a scenario, it is important to define the hypothesis being tested and what is the expected result, besides providing the evidence which will be used.

In the first scenario we have the following:

1. Hypothesis being tested
 - (a) `isShipOfInterest(ship)`
 - (b) `hasTerroristPlan(ship)`

2. Expected result

- (a) Low probability that `isShipOfInterest(ship1) = true`
 - (b) High probability that `hasTerroristPlan(ship1) = NoPlan`

3. Evidence

- (a) `hasCrewMember(ship1, person1) = true`
 - (b) `hasCrewMember(ship1, person2) = true`
 - (c) `hasResponsiveRadio(ship1) = true`
 - (d) `hasResponsiveAIS(ship1) = true`
 - (e) `hasTypeOfShip(ship1) = Merchant`

Figure 5.24 presents the SSBN network generated from scenario 1 and as expected the probability of `isShipOfInterest(ship1) = true` is low and `hasTerroristPlan(ship1) = NoPlan` is high, 1.65% and 99.96%, respectively.

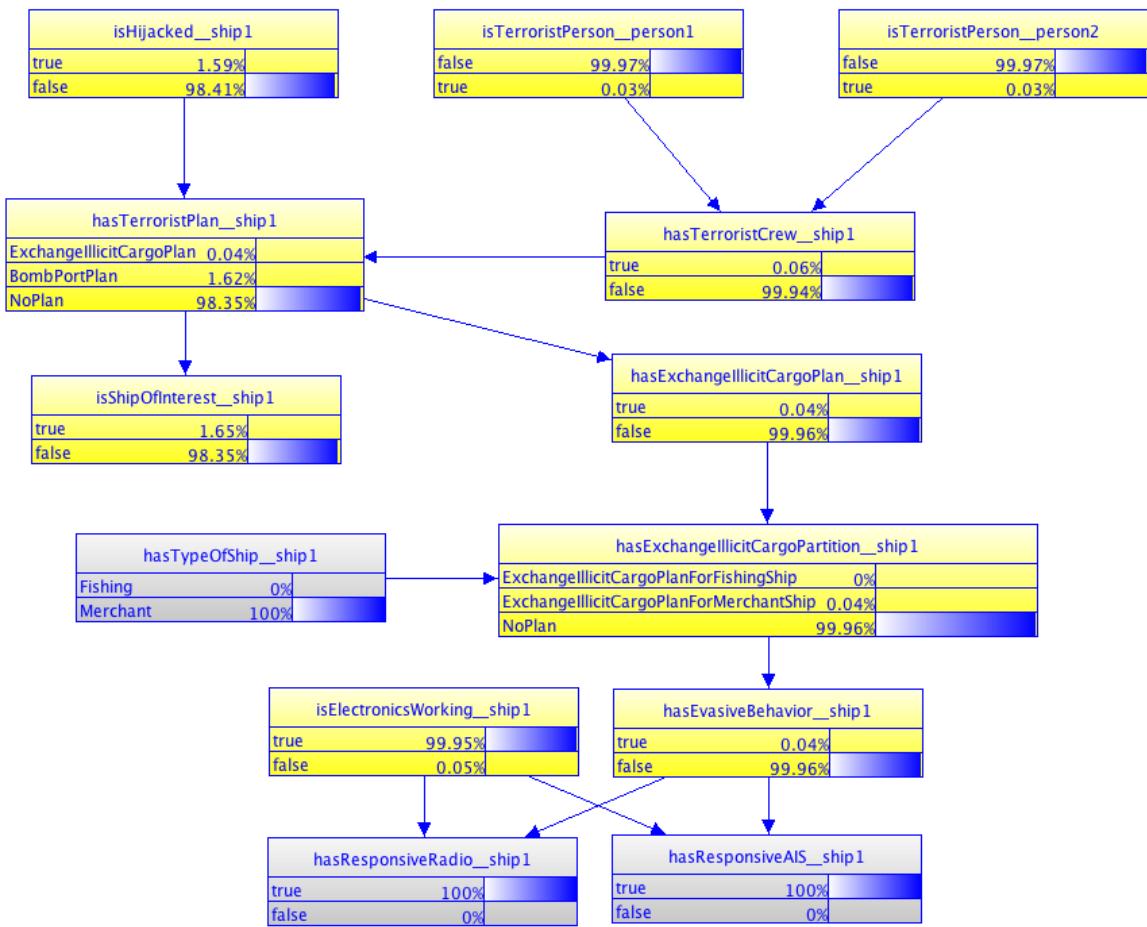


Figure 5.24: SSBN generated for the first scenario.

In the second scenario we have the following:

1. Hypothesis being tested
 - (a) `isShipOfInterest(ship)`
 - (b) `hasTerroristPlan(ship)`
2. Expected result
 - (a) Probability that `isShipOfInterest(ship1) = true` between 33% and 67%
 - (b) Probability that `hasTerroristPlan(ship1) = NoPlan` between 33% and 67%

3. Evidence (in italic we have the different evidence compared to scenario 1)

- (a) `hasCrewMember(ship1, person1) = true`
- (b) `hasCrewMember(ship1, person2) = true`
- (c) `hasResponsiveRadio(ship1) = false`
- (d) `hasResponsiveAIS(ship1) = true`
- (e) `hasTypeOfShip(ship1) = Merchant`
- (f) `hasUnusualRouteReport(ship1) = true`

Figure 5.25 presents part of the SSBN network generated from scenario 2 and as expected the probability of both `isShipOfInterest(ship1) = true` and `hasTerroristPlan(ship1) = NoPlan` are 44.27% and 58.60%, respectively.

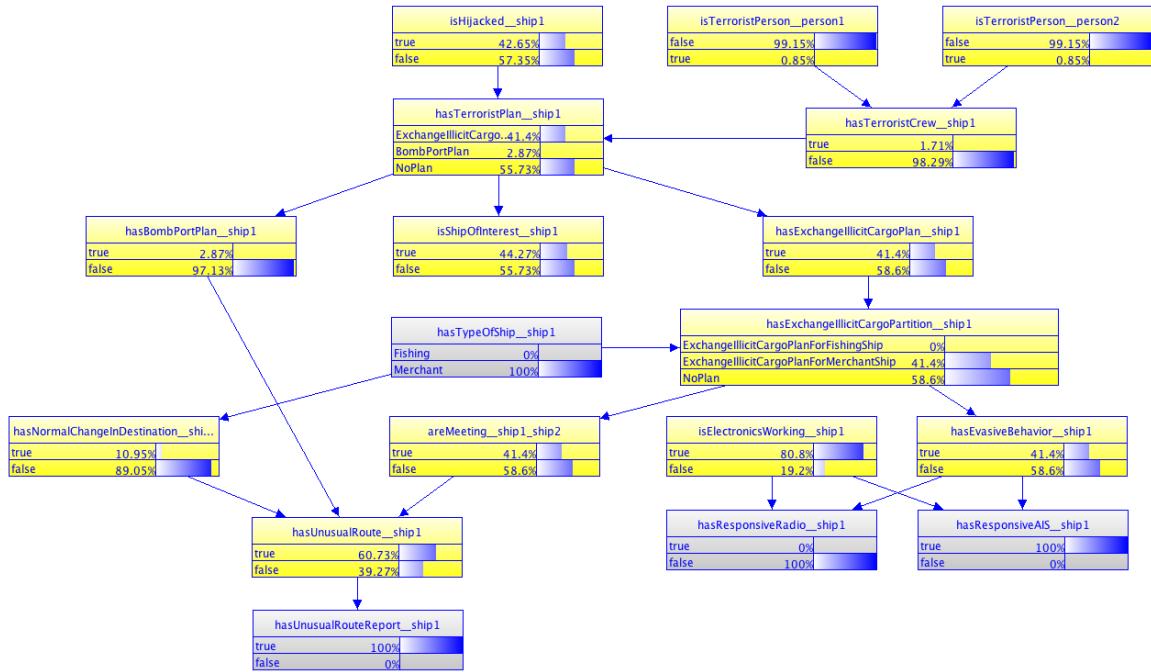


Figure 5.25: SSBN generated for the second scenario.

In the third scenario we have the following:

1. Hypothesis being tested
 - (a) `isShipOfInterest(ship)`
 - (b) `hasTerroristPlan(ship)`
2. Expected result
 - (a) Probability that `isShipOfInterest(ship1) = true` greater than 50%
 - (b) Probability that `hasTerroristPlan(ship1) = ExchangeIllicitCargoPlan`
greater than 50%
3. Evidence (in italic we have the different evidence compared to scenario 2)
 - (a) `hasCrewMember(ship1, person1) = true`
 - (b) `hasCrewMember(ship1, person2) = true`
 - (c) `hasResponsiveRadio(ship1) = false`
 - (d) `hasResponsiveAIS(ship1) = true`
 - (e) `hasTypeOfShip(ship1) = Merchant`
 - (f) `hasUnusualRouteReport(ship1) = true`
 - (g) `areMeeting(ship1, ship2) = true`
 - (h) `isJettisoningCargo(ship1) = true`

Figure 5.26 presents the SSBN network generated from scenario 3 and as expected the probability of both `isShipOfInterest(ship1) = true` and `hasTerroristPlan(ship1) = ExchangeIllicitCargoPlan` are 94.44% and 93.00%, respectively.

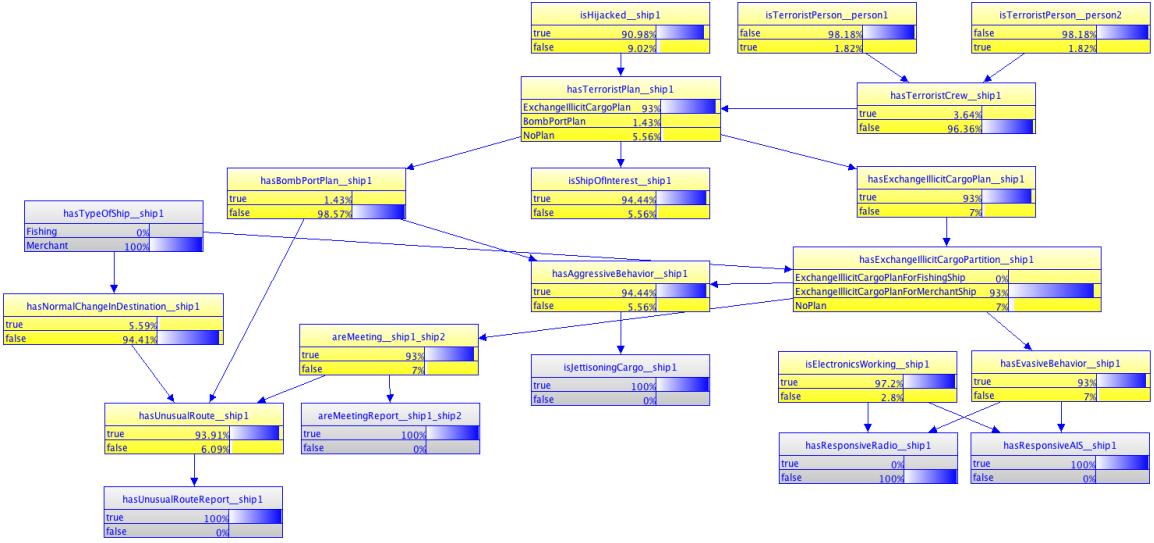


Figure 5.26: SSBN generated for the third scenario.

5.2.3 Third Iteration

While the original model considered whether a person is related to a terrorist or is part of a terrorist organization, this iteration focuses on determining whether a person *is* a terrorist. Ethical aspects excluded, creating a profile of a terrorist from the available merchant population reduces the volume of individuals requiring further investigation by limited analytic resources. The idea is to infer an individual crew member's terrorist affiliation given his close relations, group associations, communications, and background influences. The work presented in this Subsection is based on the work of Haberlin and Costa [55], which depicts a BN for this domain, and Carvalho *et al.* [18], which defines a probabilistic ontology based on the BN model from [55].

Literature on the subject reveals several models that sought to map the terrorist social network using social network analysis and some method of probabilistic inference. Using automation to identify interconnections between terrorists can reduce operator workload. Yang and Ng constructed a social network from weblog data gathered through topic-specific exploration [135]. Similarly, Coffman and Marcus performed social network analysis through

pattern analysis to classify the roles of actors within a network using communication data [25]. Dombroski and Carley propose a hierarchical Bayesian inference model to produce a representation of a network’s structure and the accuracy of informants [38]. Krebs has mapped a terrorist network topology from open-sources following the 9/11/2001 attacks and introduced a model representing the degrees of separation in Al Quaida leadership [73]. In a few cases, these network analyses were taken a step further and used to evaluate effects of friendly force courses of action, effects of removing particular individuals, and predicting attacks based on patterns of activity. Wagenhals and Levis used a timed influence net to add a temporal component to a model with terrorists embedded in a society that is supporting them to describe desired and undesired effects to both the adversary and local population caused by friendly forces [132]. Moon and Carley linked social and spatial relations to predict the evolution of a terrorist network over time, and posit the effect of “isolating” particular individuals within the network [95].

These models all concern groups, their members, and linkages. Our third iteration has the goal of applying high-level fusion by combining information about relations, group affiliations, communications, and ethno-religious or political background into a model describing the likelihood that a particular individual becomes a terrorist. This extends the overall high-level fusion MDA PO developed so far.

Requirements

The main goal is to identify the likelihood of a particular crew member being a terrorist. Specific statistics were not available in open-source material so the model assumes 0.001 percent of the target demographic to be involved in terrorism, and expands the query “Does the ship have a terrorist crew member?” as follows (same typing convention applies):

1. *Does the ship have a terrorist crew member?*
 - (a) *Is the crew member associated with any terrorist organization.*
 - (b) Has the crew member been negatively influenced in some way by his/her personal

history?

- i. Verify if the crew member has direct knowledge of someone either detained or killed by coalition forces during the conflict;
 - ii. Verify if the crew member is married.
- (c) Has the crew member been using communications media frequently used by terrorists?
- i. Verify if the crew members uses cellular communication;
 - ii. Verify if the crew members uses chat room communication;
 - iii. Verify if the crew members uses email communication;
 - iv. Verify if the crew members uses weblog communication;
- (d) Is the crew member a potential terrorist recruit?
- i. *Verify if the crew member is related to any terrorist;*
 - ii. Verify if the crew member has friendship with any terrorist.
- (e) Is the crew member associated with any of the four primary terrorist cliques introduced by Sageman who are operating in the Middle East, North Africa and Southeastern Asia [116]?
- i. Verify if the crew member is a professional, semiskilled, or unskilled laborer;
 - ii. Verify the education level of the crew member;
 - iii. Verify if the crew member is from the upper, middle, or lower class;
 - iv. Verify the nationality of the crew member.

Analysis and Design

This stage formally defines the model semantics captured in the UML model. Table 5.4 presents a two step approach to identifying the major entities, their attributes, and relationships. Initially, the requirements are the main source for keywords representing concepts to be defined in the ontology (*e.g.*, highlighted text in Table 5.4). Then, the chosen keywords are grouped in a logical manner, *e.g.*, grouping attributes with the entities possessing them

Table 5.4: A simple method for identifying entities, attributes, and relationships.

... Does the ship have a terrorist crew member ? ... Is the crew member associated with any terrorist organization Verify if the crew member is married Verify if the crew members uses cellular communication ; ...	Ship -hasCrewMember Person -isMemberOfOrganization -isMarried -usesCellularCommunication
--	---

(see simple grouping on the second column). Although not shown here for brevity, this method was used for the analysis and design of all the requirements in this iteration. The resulting attributes, relationships and their grouping for the entities **Person** and **Organization** is shown in Table 5.5.

These three iterations are meant to illustrate the probabilistic definitions of the ontology, and thus reflect just the initial steps in building a full model. Further analysis of the terms listed in Table 5.5 will show that other entities are necessary to encode the MDA complete semantics. For instance, the **Country** entity is needed to express the relationship that **Person hasNationaly some Country**. The next step is to understand the domain rules, making use of the concepts identified so far to achieve the goals elicited during the requirements stage. The following rules, already grouped in fragments, were identified after a review of the open source literature available (same typing convention applies):

1. Terrorist organization grouping;
 - (a) *If a crew member is a member of a terrorist organization, then it is more likely that he is a terrorist;*
 - (b) *If an organization has a terrorist member, it is more likely that it is a terrorist organization.*
2. Background influence grouping;

- (a) For those who are terrorists, 100% of them chose to do so because of something in their past. That is, no one was born a terrorist, or just woke up one day and decided to be a terrorist. That is the easy case. For those who are not, 20% chose not to become terrorists despite having some possible factor in their background and 80% chose not to become a terrorist possibly because they have never been exposed⁷.
- (b) An individual is usually negatively affected (leads him/her in becoming a terrorist) by having direct knowledge of someone either detained or killed by coalition forces during the conflict;
- (c) In the geographic area of interest, an estimated 2% of the population knows someone who was killed as a result of OEF/OIF [94];
- (d) In the geographic area of interest, approximately 2% of the population knows someone detained as a result of coalition operations [94];
- (e) Contrary to common perception, terrorists are predominantly married in keeping with the teachings of the Quran [116]. And about half of the general population in the target demographic is married.

3. Communication grouping:

- (a) It is possible that a crew member may communicate with a terrorist without being involved in terrorism due to non-terrorist affiliations or other relationships that have some normal expectation of interaction;
- (b) For each of the internet communications paths there is also the background usage rate of 28.8% in the Middle East [5]. Because the data is not broken down for the three internet transmission paths, this probability was applied equally to chat room, email, and weblog methods of communication;
- (c) Similarly, cellular telephone usage among the general population is assumed to be 31.6% based on Egyptian subscriber rates [4];

⁷This rule and explanation was given by the SME.

- (d) Given the availability of cellular technology and subscribing to the prioritization, a probability of 90% is assigned to terrorists communicating using cellular telephones;
- (e) The transient nature and unfettered availability of chat room communications makes it appealing to individuals who desire to remain nameless. A probability of 85% is assigned to terrorists communicating through chat rooms;
- (f) Email is the least desirable form of communication because it requires some form of subscriber account. Even in the event that fictitious information is used in creating such an account, an auditable trail may lead determined forces to the originator. Still, it is a versatile means of communication and is assigned a probability of 65% for terrorists;
- (g) The anonymity associated with weblog interaction is very appealing to terrorists. This path is similar to chat room communications, but is less transient in content and can reach more subscribers simultaneously. For these reasons, a probability of 80% is assigned to weblog communications.

4. Relationship grouping;

- (a) Research shows that if a crew member has a relationship with terrorists, there is a 68% chance that he has a friend who is a terrorist;
- (b) Research shows that if a crew member has a relationship with terrorists, there is a 14% chance that he is related to a terrorist.

5. Cluster grouping;

- (a) It is assumed that all active terrorists fall into one of the terrorist cliques or their subsidiaries described by Sageman [116];
- (b) Contrary to popular thought, terrorists tend to not be unskilled drifters with no options other than martyrdom;

Table 5.5: Grouping for entities, attributes, and relations in third iteration.

Terrorist grouping -Person -isTerrorist -Organization -isMemberOfOrganization -isTerroristOrganization	-hasFriendshipWithTerrorist
Communication grouping -Person -usesWeblog -usesEmail -usesCellular -usesChatroom	Background influence grouping -Person -hasInfluencePartition -hasFamilyStatus -hasOIForOEFInfluence -knowsPersonKilledInOIForOEF -knowsPersonImprisonedInOIForOEF
Relationship grouping -Person -hasTerroristBeliefs -hasKinshipToTerrorist	Cluster grouping -Person -hasClusterPartition -hasNationality -hasEconomicStanding -hasEducationLevel -hasOccupation

- (c) Many believe terrorist recruits to be uneducated simpletons who are easily persuaded by eloquent muftis who appeal to their sense of honor and perception of persecution. In fact, the data indicate that the typical terrorist is more educated than the average global citizen and is by far more educated than the average citizen in the Middle East, North Africa, and Southeastern Asia regions [116];
- (d) Terrorist from the clusters described by Sageman [116] are less likely to be of lower class than other people from that demographic area.

Given the extensive research previously done, it was possible to assert some probability values when elaborating these rules during the Analysis & Design stage, whereas in previous iterations probabilities were defined only during the Implementation stage. Usually, only imprecise statements are used in these conditional rules (*e.g.*, more likely, less likely, rare, etc).

Implementation

Appropriate assumptions are needed to accommodate available data without compromising the utility of the model. First, a terrorist will communicate with other terrorists with certainty, but there is variability on the communication path used. Also, an individual might communicate with terrorists inadvertently. Next, there is 0.1% chance that any random person in the target demographic is a terrorist, which drives the coincidental interaction between a honest crew member and someone who may happen to be a terrorist without his knowledge. Further, the target area (Middle East, North Africa and Southeast Asia) enables using the cluster organizations introduced by Sageman [116] as basis for the groups in the association partition. Other attributes within this partition are compiled given the individual's participation in one of those groups. Additionally, a crew member could be involved with a terrorist organization other than the four identified, and that would negatively affect the outcome since he would be grouped with non-terrorists. However, it is likely that smaller groups are splinters from one of these major clusters and could therefore be included in the analysis under their super-group. Finally, in its current form, the model only captures the influence of Operation Enduring Freedom (OEF) and Operation Iraqi Freedom (OIF) and marital status in the crew member's background. Figure 5.27 presents the last MFrags changed/added to the MTheory for Domain Maritime Awareness (see Figure 5.23 for previous MFrags).

Appendix B Subsection B.2.3 presents the details and explanations of all MFrags and all resident nodes and their respective LPDs of the probabilistic ontology discussed in this Subsection.

Test

Again, although I have described many different types of evaluation and tests we can perform in our model in Subsection 5.1.4, this iteration will focus on performing integration test based on case-based evaluation, as was the case in the first and second iterations.

As explained in Subsection 5.1.4 it is important to try out different scenarios in order to

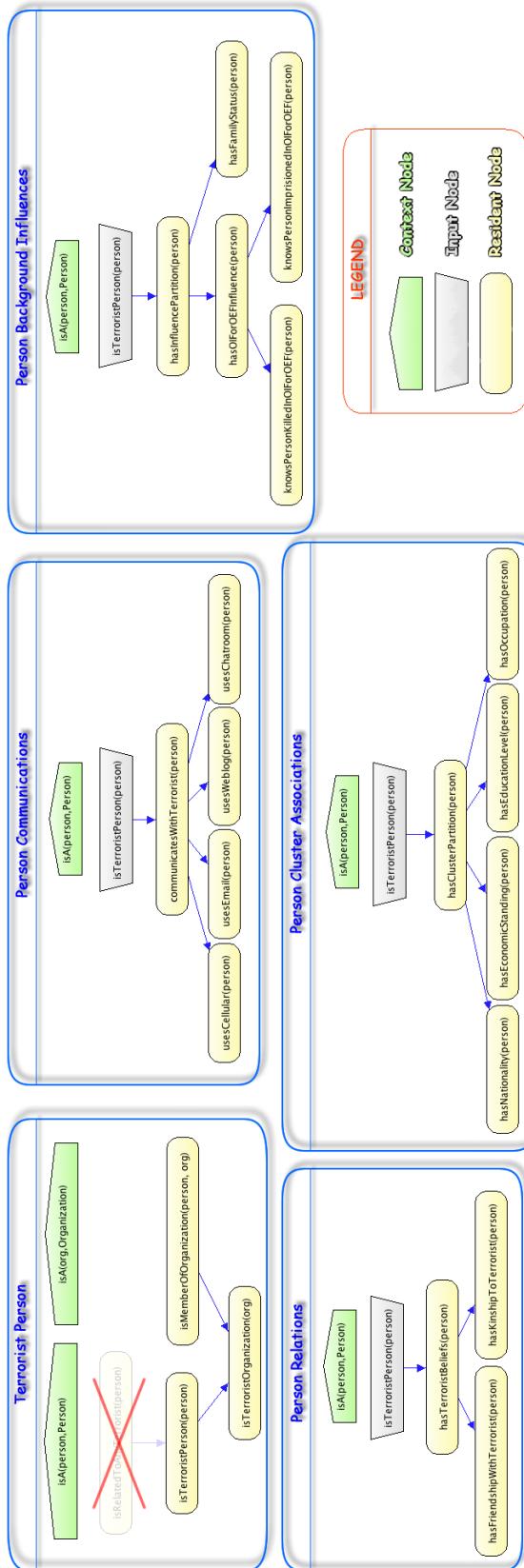


Figure 5.27: MFrags changed/added in third iteration.

capture the nuances of the model. In a serious test of the model, we would have to model a lot scenarios in order to cover at least the most important aspects of our requirements. However, I define only three qualitatively different scenarios in order to illustrate the mechanics of defining and testing a scenario. The first is a general case in which an individual fits a profile and can therefore be “correctly” identified. In the second and third cases, situations are introduced in which individuals could be incorrectly profiled using these techniques.

In the first scenario (“obvious” guilty), Bakari, a student at Misr University in Cairo and member of a terrorist organization, has been tasked with smuggling explosive materials into the United States for use in making improvised explosive devices (IED). He is from a middle-class Egyptian family with a large extended family, including one uncle who is a member of the Mojahedin-e Khalq Organization. Because he is a full-time student, he has not had the opportunity to earn enough money for a suitable dowry and is still single. Recently, postings on a terrorist-related weblog have been attributed to Bakari’s school account, in which he laments his colleagues he watched being taken prisoner by the coalition.

Figure 5.28 shows that with all the information above the probability that Bakari is involved in terrorism is 72.59%, primarily due to the weblog communications and affiliation of his uncle. Removing the communications link drops him all the way down to 48.85%. Including communications activity and removing the uncle affiliation drops his percentage to 3.07%. It is clear that being related to and communicating with terrorists will flag an individual very significantly as a terrorist candidate.

These results are taken into consideration in new iterations. Usually, LPDs are adjusted in order to make probabilities that were too high go down, and vice-versa. Although it is not described here, this was done a few times during this modeling process. In fact, before this model can be deployed, it should go through a few more iterations of adjusting the distributions.

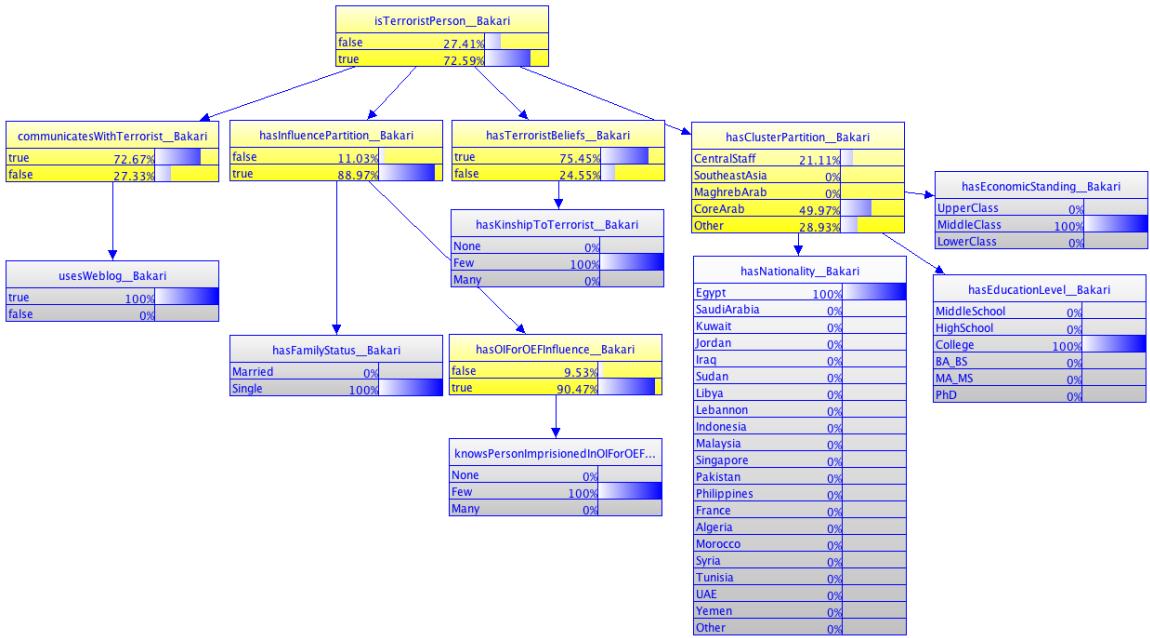


Figure 5.28: SSBN generated for scenario 1.

Also of note is effect of the Influence Partition on the outcome of this case. The scenario introduced information about Bakari's marital status, and this has very little effect. Removing the marital status results in a probability of being a terrorist of 76.27%. This value is higher, because the "standard" terrorist profiled requires an individual to be married, not single. Knowing someone imprisoned has a greater effect and removal of this information reduces the overall terrorism likelihood to 35.33%.

It is clear from this case study that family and friend relationships weigh heavily on the determination of terrorist activity. In the case where an individual has a casual or coincidental relationship with someone involved, or there is a case of name-based mistaken identity, this would likely lead to an incorrect determination. Ranking the partitions from most influential to least gives an ordering of Relationship, Communication, Influence, and Cluster.

In the second scenario (guilty who looks innocent), Arif leaves his Indonesian village at age 17 to provide for his family through life as a merchant sailor. He is an unmarried,

unskilled worker who did not complete high school. While looking for work as a mariner in Jakarta, he shared a room with 5 others, at least one of whom has become involved with the Jemaah Islamiyah organization. Arif joins his friend at a Jemaah Islamiyah meeting where he is given a cell phone and contact information.

In this case, Arif is involved in the beginning stages of the terrorist recruitment process. While his background has none of the profile indicators, his growing affiliation and recruitment will eventually lead him to a positive assessment. It is nearly impossible to force a positive likelihood onto the crew member being a terrorist by switching features in the Influence, Communications, and Relationship partitions. This is due to the fact that the cluster partition has driven the model to an unlikely terrorist character in the “Other” category (see Figure 5.29). Since he does not fall into one of the terrorist cliques, it will be difficult to identify him as a terrorist. His background does not fit with the classic profile.

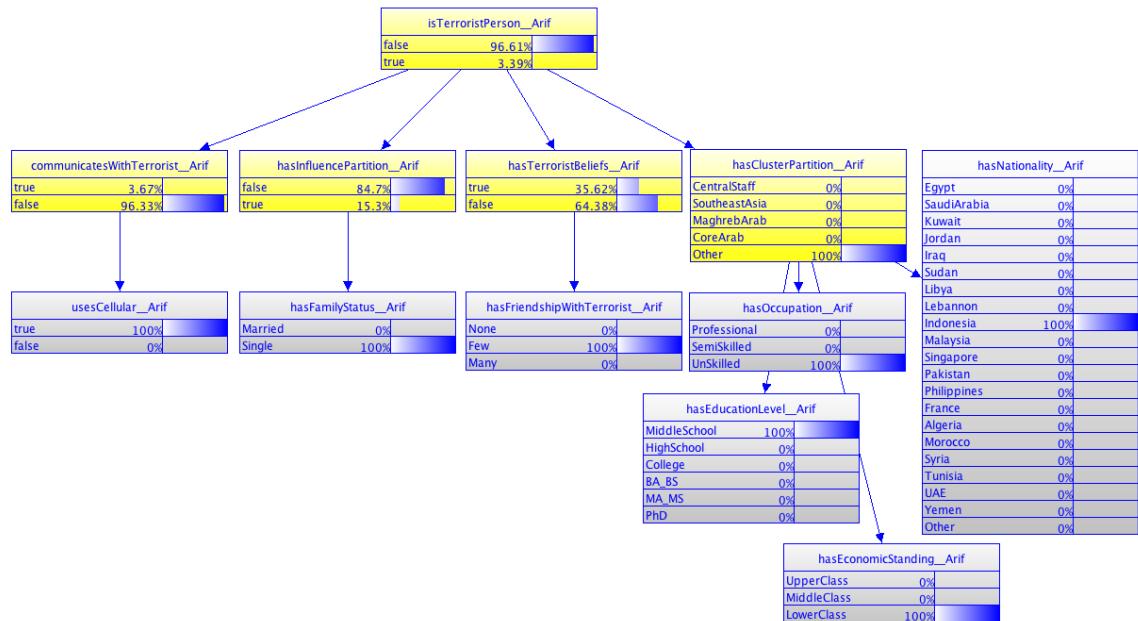


Figure 5.29: SSBN generated for scenario 2.

This scenario demonstrates a weakness of the model and intelligence collection in general.

Profiles are built on history, but cannot account for rapid transition from one social group to another. Arif arrived in Jakarta as a farmer looking for work and through rapid social affiliation became a terrorist suspect. The unknown question is whether he will continue to grow his relationship with Jemaah Islamiyah, or turn toward life as a commercial seaman.

In the third scenario (innocent who looks guilty), Irasto leaves Amman, Jordan to earn a living as a merchant sailor. He comes from a middle class family and began studies at the University of Jordan before local violence frightened him into leaving. While in school, several of his friends were detained by coalition forces under suspicion of terrorist affiliation and have not been seen since. He frequently communicated with them by email and cell phone prior to their disappearance.

The unknown status of Irasto's friends muddies the waters for this scenario. They were detained as part of OEF/OIF, and therefore affect the Influence Partition, but we have no information as to whether these friends were actually confirmed to be terrorists. If the safe route is taken (from an intelligence perspective) they will be considered terrorists and Irasto will also be pronounced a terrorist with a likelihood of 89.92%; without this assumption the probability drops to 3.44%. These are the worst and best case, respectively. However, if we considered the likelihood that his friends are terrorists, then we would obtain a probability between those two numbers (the extreme cases).

In this iteration we are simplifying this friendship relationship. In fact, `hasFriendship-WithTerrorist` is logically equivalent to the existentially quantified RV saying there exists x such that x is a terrorist and x is friends with Irasto. This RV is a built-in RV in MEBN. However, due to UnBBayes limitation, we are considering this existential operation is done outside the model and we just receive the result (`None`, `Few`, or `Many`). The problem with this approach is that if we want to infer the likelihood, for instance, that Irasto's friends are terrorists using our model, their probability will not influence Irasto's probability of being a terrorist, since there is no connection between the node `hasFriendshipWithTerrorist` and Irasto's actual friends. In future iterations this has to be dealt with.

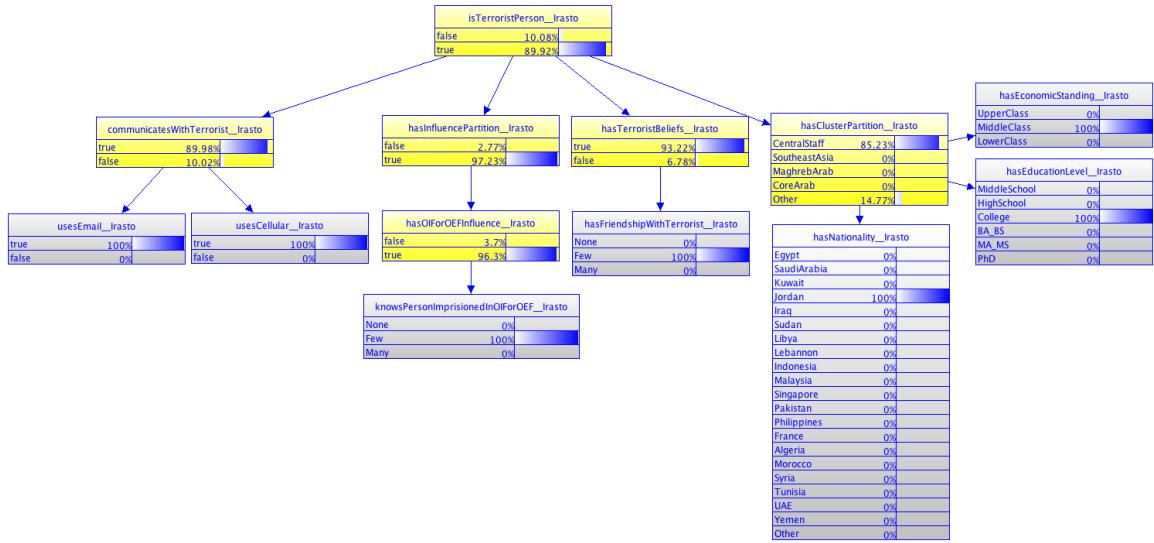


Figure 5.30: SSBN generated for scenario 3.

The model indicates that Irasto appears to be involved with either the Central Staff or Maghreb Arab clique. This drives the Relationship partition into strongly affecting the overall likelihood. The same dilemma exists for communications. Irasto communicated with his friends using two of the profiled communication paths. If those friends are determined to be terrorists, then his likelihood jump significantly over what it would be if they are not. The model recognizes guilt by association. These two particulars illustrate some of the problems introduced when intelligence is not shared between organizations. If the analysts have access to the final determination of his friends, Irasto will be more likely to have a correct determination of guilt or innocence.

5.2.4 Testing the Final MDA PO

One of the major challenges in systems like PROGNOS is evaluating the situational awareness and prediction generated by its probabilistic model.

The literature in Machine Learning, Artificial Intelligence, and Data Mining usually work with real data by separating it into training and testing sets. However, in systems

that try to predict rare events, such as terrorist attacks, either there is not enough data or the data available is classified. Therefore, in these cases, there is not sufficient data to be used for testing.

To overcome this limitation, a common approach is to create different use cases or scenarios manually. This use case generation process is discussed in Subsection 5.2.4. However, this is a tedious process and usually not statistically sufficient to confidently assess how good these probabilistic models are. In Subsection 5.2.4 we present a framework that can be used for automatically creating different and random, yet consistent, scenarios to provide sufficient statistical data for testing. It is to be stressed, however, that this testing is only as good as the use cases incorporated into the testing process, and there is no substitute for real-world evaluation. It is important to notice that although this Subsection focuses on case-based evaluation, this should not be the only test done in the final model. In fact, Subsection 5.1.4 presents all the tests that should be performed in the Test discipline.

Creating scenarios manually

In the first iteration the main goal is to identify if a ship is of interest, *i.e.*, if the ship seems to be suspicious in any way. The assumption in this model is that a ship is of interest if and only if there is at least one terrorist crew member.

The following iterations provide clarification on the reasons behind declaring a ship as being of interest and detects an individual crew member's terrorist affiliation given his close relations, group associations, communications, and background influences.

To test this final probabilistic model, let's define 4 major scenarios:

1. a possible bomb plan using fishing ship;
2. a possible bomb plan using merchant ship;
3. a possible exchange illicit cargo using fishing ship;
4. a possible exchange illicit cargo using merchant ship.

For each of these major scenarios let's create 5 variations:

1. “sure” positive;
2. “looks” positive;
3. unsure;
4. “looks” negative;
5. “sure” negative.

All 20 different scenarios were analysed by the SMEs and were evaluated as reasonable results (what was expected). Figure 5.31 presents part of the SSBN generated for a scenario where a merchant ship is exchanging illicit cargo and the evidence makes it obvious to detect that this is the case. In order to be concise, and since the focus is on the automatic testing presented in Subsection 5.2.4, this will be the only scenario presented.

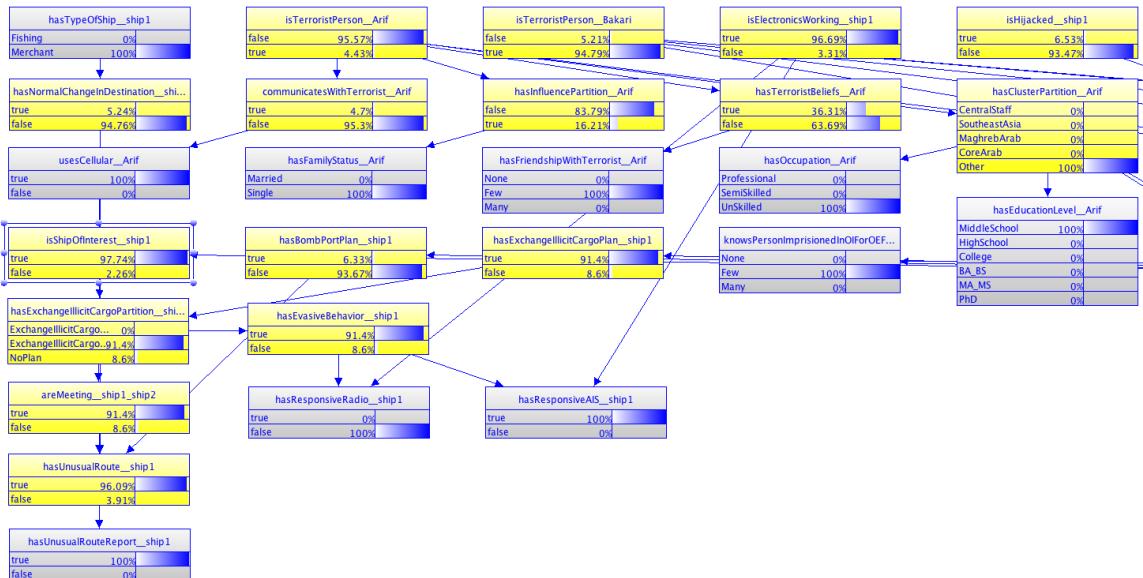


Figure 5.31: Part of the SSBN generated for “sure” positive of a possible exchange illicit cargo using merchant ship.

Creating scenarios automatically

Besides being a tedious process, there are a few problems with the manual creation of scenarios as presented in Subsection 5.2.4 and in the scenarios tested for each individual iteration. In the first set of scenarios created for the first iteration, it is clear that the test designers just tested how well the model behaves when all the evidence is in favor of the hypotheses being tested. However, how will the model behave if we receive evidence both in favor of and against the hypotheses being tested? Is it still a good model in these cases?

In fact, this is a problem that the last set of scenarios presented in Subsection 5.2.4 addresses. This, the fact that some evidence favors the hypotheses and some does not, is why there are scenarios where the expected result is “looks” positive, “looks” negative, and unsure. However, even twenty different scenarios is not enough considering the amount of information that is used as evidence in the final model. Let’s clarify by presenting the numbers. In the final model there are more than 20 evidence nodes with at least 2 states each (some have more than 10 states). This gives more than $2^{20} = 1,048,576$ different configurations of evidence. In other words, while we tried to cover different types of scenarios, 20 is still an extremely small number compared with the possible configurations. However, it is unreasonable to think a human being will be able to generate and analyze more than one million different scenarios. For this reason, we created a framework for simulating different scenarios automatically for the PROGNOS project [28].

There are three basic steps in our simulation framework:

1. Create entities and generate some basic static ground truth for them (*e.g.*, create ships, define their type, and their plan);
2. Generate dynamic data for entities based on their static ground truth data (*e.g.*, if the ship is a fishing ship and it has a normal behavior, it will go from its origin port to its fishing area and after some time it will go to its destination port);
3. Simulate reports for different agencies. Each agency has a probability of generating a correct report (*e.g.*, saying a person is from Egypt when he is actually from Egypt),

an incorrect report (*e.g.*, saying a person is not from Egypt when he is in fact from Egypt), and no report at all (*e.g.*, not being able to say where a person is from). The idea is that different agencies are expected to be more accurate in certain types of information than others (*e.g.*, the Navy is expected to have more accurate data on a ship’s position than the FBI).

The information generated in the first two steps are considered the ground truth, while the reports generated in the third step is given as input to the probabilistic model, like the MDA PO described in this Section. The probabilistic model can then use these information as evidence to provide situational awareness and prediction after the reasoning process through its posterior probabilities. Once we know what the model “thinks” is more reasonable (*e.g.*, if a ship is of interest), we can ask the simulation for the correct information, *i.e.*, the ground truth with respect to the hypotheses being tested (*e.g.*, if the ship is indeed of interest). We can then evaluate if the model provided a correct result. Since this process is automatic, we can run this evaluation process as many times as we need to and finally compute some metrics (*e.g.*, confusion matrix) to evaluate how well our model performs. Furthermore, a subset of these generated scenarios should be selected in order to be presented to the SMEs to determine whether the results are reasonable.

Table 5.6: Number of TP, FP, TN, and FN.

Real/Inferred	T	F
T	24	3
F	11	577

To test the final MDA PO, I ran the simulation with 615 ships, where 27 of them were ship of interest and 588 were regular ships with no terrorist plan. Tables 5.6 and 5.7 present the confusion matrix for the node `isShipOfInterest(ship)`. Table 5.6 presents the

number of ships while Table 5.7 presents the probability of true positive (TP), false positive (FP), true negative (TN), and false negative (FN). As It can be seen, the percentage of misclassifications of ships of interest was small, only 3 in 27, *i.e.*, only 11.11%.

Table 5.7: Percentage of TP, FP, TN, and FN.

Real/Inferred	T	F
T	88.89%	11.11%
F	1.87%	98.13%

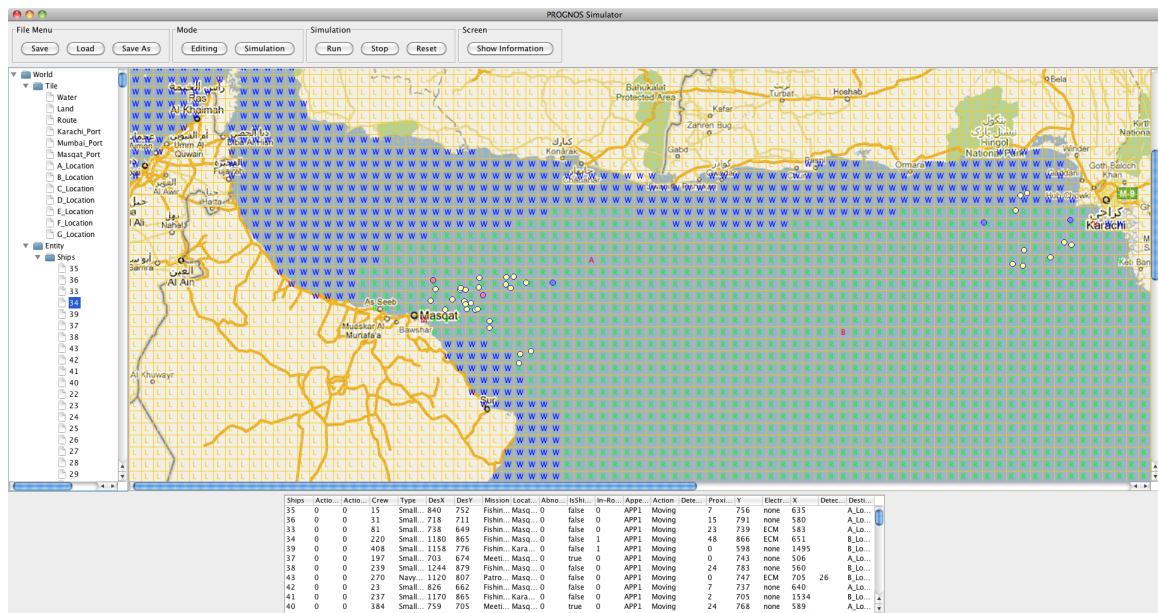


Figure 5.32: Simulation editor.

In the case of the PROGNOS evaluation, the subject matter experts who evaluated the use cases also supported the domain knowledge engineering effort. A more rigorous evaluation process would make use of independent subject matter experts who had not

been involved in the system design process. These independent evaluators would develop use cases for evaluation and rate the quality of the system's output.

However, to be able to compute the three steps described above, we need to define some basic characteristics of the simulation. For instance, what is the geographical region considered, which cells correspond to land and which correspond to sea, where are the ports of interest, what are the usual routes between areas of interest, where are the common fishing areas, etc. Figure 5.32 presents the simulation editor used to define this information.

Chapter 6: Conclusion

The main contribution of this research is two-fold: to enhance the syntax and semantics of PR-OWL in order to attaining full compatibility between the Web Ontology Language (OWL) and the Probabilistic Web Ontology Language (PR-OWL); and to propose a methodology that describes how to model a probabilistic ontology and to use it for plausible reasoning.

Chapters 2 and 3 presented the current research on areas of knowledge modeling by describing different approaches and how they differ from the perspective of achieving the full potential of the Semantic Web, and on areas of logic and ontology languages by describing various extensions to First-Order Logic (FOL) and ontology languages for allowing the representation of uncertainty.

The enhanced syntax and semantics of PR-OWL was described in Chapter 4, where a formal mapping between OWL concepts and PR-OWL random variables is described in order to address the problem of attaining full compatibility between OWL and PR-OWL. This new syntax and semantics is defined as PR-OWL 2.

First, the importance of a formal mapping was justified through an example. Second, a simple solution sufficient for 2-way relations was presented. Then, a more complex and robust solution covering n-ary random variables was presented. Finally, a schematic was given for how to do the mapping back and forth between PR-OWL random variables and OWL triples (both predicates and functions).

Poole *et. al.* [110] emphasizes that it is not clear how to match the formalization of random variables from probabilistic theories with the concepts of individuals, classes and properties from current ontological languages like OWL. However, Poole *et. al.* [110] says that “We can reconcile these views by having properties of individuals correspond to

random variables.” This is the approach used in this work to integrate MEBN logic and OWL language.

Furthermore, PR-OWL 2 also adds compatibility to OWL’s primitive types and it was shown that this plays an important role when trying to define the distribution of continuous random variables, for instance.

One of the major changes presented in PR-OWL 2 is that there is a separate class to define random variables, and nodes make reference to this class. This provides a series of benefits:

- It is possible to decide dynamically which distribution to use. This is a form of polymorphism that can be very useful to the modeler.
- As described in MEBN, every random variable should have a unique default distribution. It is now possible to guarantee this constraint in PR-OWL 2.
- In PR-OWL 2 the mapping between a PR-OWL random variable and an OWL property is defined only once. If the resident node was used instead for the mapping, the same mapping would eventually have to be defined more than once.

Since PR-OWL 2 now makes use of OWL’s class structure, all the control over the type definition and type hierarchy in PR-OWL 2 is now delegated to OWL. This was not the case in PR-OWL 1.

Besides the built-in logical operators and quantifiers, PR-OWL 2 also provides a built-in random variable for defining type uncertainty, which is already mapped to RDF’s type property.

The community is already familiar with the types of reasoning available for the semantic web when using logic-based languages like OWL. So a common question is: “What types of reasoning can one expect from probabilistic languages for the semantic web?”

According to Poole *et al.* [110], when integrating probability theories and ontological languages three types of uncertainty reasoning are expected: existential uncertainty; type uncertainty; and property value uncertainty.

However, these are not the only ones. For instance, Costa [27] talks about identity uncertainty and association uncertainty (also known as reference uncertainty [47]). Nevertheless, they can also be easily reduced to property value uncertainty. Therefore, property value uncertainty plays the same crucial role in probabilistic first-order language reasoning as satisfiability does for logical reasoning.

Thus, it is important to notice that PR-OWL 2 besides integrating well with the web ontology language (OWL) by mapping random variables to properties, it supports all major uncertainty reasoning expected from a probabilistic first-order language.

The new syntax and semantics is not only described, but a series of examples are also presented in Chapter 4 to illustrate its use and that it covers all possible definitions from the MEBN logic.

The lack of support in probabilistic ontology engineering is addressed in Chapter 5 where the UMP-ST methodology for modeling probabilistic ontologies is described. Two different models were designed and described step-by-step from scratch in order to illustrate how the methodology works and to verify it can be applied to different scenarios. One model is used for identifying frauds in public procurements in Brazil¹ and the other is used for identifying terrorist threats on the maritime domain². In both use cases it was highlighted PR-OWL's proposed version, PR-OWL 2, advantages compared to PR-OWL 1.

On the one hand, Section 5.1 focused on presenting in detail the activities that must be executed in each discipline in the POMC cycle. On the other hand, Section 5.2 focused on presenting how the model evolves through time with every new iteration.

The objective of the first was to present as much detail as possible on the steps necessary to model a probabilistic ontology using the POMC cycle. The objective of the second was to show that the UMP-ST process provides a useful approach for allowing the natural evolution of the model through different iterations.

¹This use case has been developed with the support of the Brazilian Office of the Comptroller General (CGU), which has been providing the human resources and the information necessary to conduct this research since 2008.

²This use case was developed as part of the PROGNOS project [32], which has been partially supported by the Office of Naval Research (ONR), under Contract#: N00173-09-C-4008.

In summary, this research provides the following contributions:

1. A formal and extended definition of PR-OWL including the connection between a statement in PR-OWL and a statement in OWL.
2. PR-OWL 2 syntax - an upper-ontology that captures the formal definition described above.
3. PR-OWL 2 semantics - a clear specification of those aspects of the mappings from PR-OWL to OWL for which OWL has no formal semantics.
4. A proof of concept tool which allows the use of PR-OWL 2 to model probabilistic ontologies.
5. Methodology for modeling POs.
6. Use cases which use PR-OWL 2 and shows its benefits when compared to the current version of PR-OWL.

6.1 Future Work

A natural next step in this research is the implementation of PR-OWL 2. In fact, this is already being addressed by the Group of Artificial Intelligence (GIA) from the University of Brasília, Brazil [88].

The same applies to the UMP-ST process. It would be interesting to have a tool guiding the user on the steps necessary to create a probabilistic ontology and link this documentation to its implementation in UnBBayes PR-OWL 2 plug-in, for instance. A tool to help in this documentation process is also being developed by the Group of Artificial Intelligence (GIA) from the University of Brasília, Brazil [35].

Even though tools are already being developed for PR-OWL 2 and UMP-ST, in order to make sure they will be widely used by the community, two major problems should be addressed, which are strictly related to the MEBN logic. The first is scalability, which is the

trade-off for the expressive power MEBN has. This is a common problem in FOPLs. Fortunately, some general solutions to this problem were already proposed, *e.g.*, Lifted Inference [12]. Other common solution is to use approximation, *e.g.*, MC-SAT and lazy inference [39], which are used for inference in MLNs. Even though these ideas are also applicable to MEBN, there are currently no algorithms available that apply those ideas to address this scalability problem in MEBN. There have been publications on hypothesis management that also address scalability [56, 57], however, only high-level ideas are presented and there is no detailed algorithms that present a solution to this scalability problem in MEBN.

A key aspect when dealing with scalability is to actually understand the complexity of the language. Worst case complexity of MEBN is known to be undecidable, *i.e.*, some queries never terminate. This is because MEBN can represent full FOL. An interesting idea for future work would be to define PR-OWL sublanguages, as was done with OWL, and classify the sublanguages by complexity. For instance, in Section 4.2 I present a solution for mapping PR-OWL n-ary RVs to OWL properties. However, what would be the consequence, as far as complexity goes, if we define a sublanguage that is able to represent only two-argument RVs? What would be the complexity of a sublanguage that allowed only DL formulas on the context nodes and DL expressions (formulas or terms) on input nodes, instead of full FOL ones? Another important area of research is to understand the accuracy and complexity of approximate reasoning algorithms for given classes of problems. For these areas of research, work already done (*e.g.*, [70]) could be adapted to PR-OWL and MEBN.

The second major problem is learning, which is similar with scalability as far as having solutions available for other formalisms that could be adapted to MEBN. For instance, although inductive logic programming (ILP) [34] has been extended to support both the learning of concept definitions and probabilistic inclusions [86], there is no algorithm available for learning in MEBN.

An interesting extension to PR-OWL is the inclusion of RVs that can be solved by external tools, which are more suitable to solving specific problems

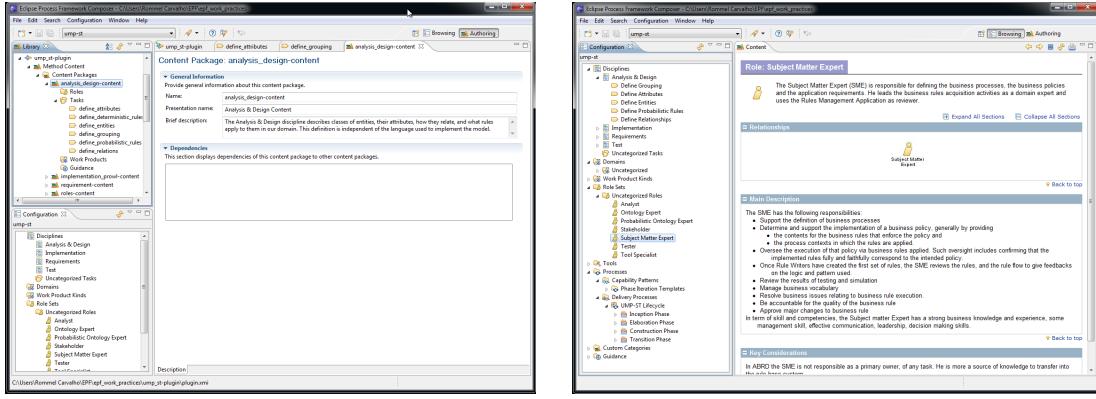
than pure MEBN logic. For instance, a common function that is used often in programming languages is the '`>`', which could be represented by the RV `greaterThan(number1,number2)`. This RV would be really useful in the procurement use case, for instance. One could say that `if priceOf(procurement) > 1,000,000.00 and annualIncome(ownerOfWinnerEnterprise) < 50,000.00 then` it is likely that `ownerOf(winnerOf(procurement))` is a front to that enterprise. This condition could be easily represented with the `greaterThan` RV and it is something that an external tool would handle much faster than using pure logic. This could be extended to define semantics for math functions like sine, cosine, and tangent and others. The external function would be called to give the value of that function given a specific argument. This would make PR-OWL much more expressive and useful while avoiding an increase in computation time and complexity, since this is easily done by external and specialized tools.

Finally, even though UMP-ST has been described in detail, there is still a lot to be addressed. For instance, there are disciplines that were not even presented in the UMP-ST, *e.g.*, configuration management and user experience design. Besides that, the UMP-ST process would benefit from having a more detailed description of the activities, roles, and artifacts involved in it.

Since UMP-ST is based in the Unified Process (UP) it could use the Eclipse Process Framework (EPF) to have a more structured way to present its disciplines, activities, best practices, roles, etc. The EPF aims at producing a customizable software process engineering framework. It has two major goals [45]:

To provide an extensible framework and exemplary tools for software process engineering - method and process authoring, library management, configuring and publishing a process.

To provide exemplary and extensible process content for a range of software development and management processes supporting iterative, agile, and incremental development, and applicable to a broad set of development platforms and applications.



(a) Using the EPF Composer tool to tailor the UMP-ST process.

(b) Example of how UMP-ST process could be defined.

Figure 6.1: Example of how to use EPF to tailor the UMP-ST process.

A process that is made freely available with the EPF framework is the OpenUP [9] which is a minimally sufficient software development process. This process could be used as a starting point to describe the UMP-ST process, since OpenUP is extensible to be used as foundation on which process content can be added or tailored as needed. Figure 6.1 shows how the EPF framework and the OpenUP could be used in order to tailor the UMP-ST process.

Appendix A: PR-OWL 2 Abstract Syntax and Semantics

This Chapter presents the Abstract Syntax and Semantics of PR-OWL 2. In all following examples I will use the prefix `pr-owl2` for the PR-OWL 2 ontology with URI “<http://www.pr-owl.org/pr-owl2.owl>” and the prefix `ex` for the example ontology with URI “<http://www.pr-owl.org/example.owl>”. The Listings are presented using the Manchester Syntax for the OWL language [64].

Figure A.1¹ presents the general hierarchy of the classes defined in PR-OWL 2.



Figure A.1: The hierarchy of PR-OWL 2 classes.

¹This hierarchy graph was generated using the OWLViz plugin for Protégé, which comes with the default distribution of Protégé 4. The homepage of the OWLViz project is <http://code.google.com/p/owlviz/>

A.1 Random Variables

A random variable, in probability and statistics, is a function that maps elements of a sample space to real numbers. The sample space represents all possible outcomes of an event or experiment. The value of the random variable is unknown before the event happens. Although the outcome is mapped to a real number, the real number can be used to represent categorical values (*e.g.*, low, medium, high), Boolean values (true and false), and other types of values different than real numbers, as long as they represent mutually exclusive and collectively exhaustive outcomes. More generally in the Artificial Intelligence (AI) literature, random variables can on values from an arbitrary set.

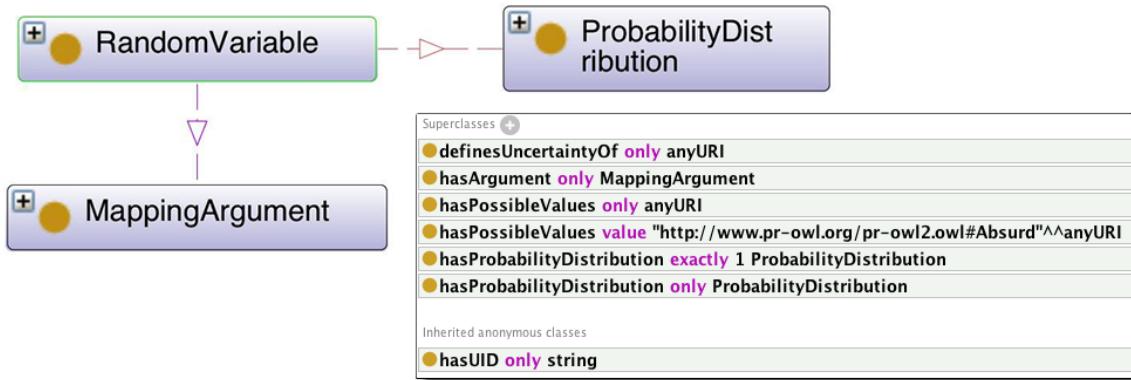


Figure A.2: The OWL restrictions of the `RandomVariable` class.

In PR-OWL 2 a random variable defines the uncertainty of the outcome related to a specific property, which has its semantics defined in OWL. There are four main concepts that need to be defined for every random variable (see Figure A.2)²: a link to the OWL property it defines the uncertainty of (represented by the property `prowl2:definesUncertaintyOf`); the domain of the random variable defined by its arguments (represented by the property `prowl2:hasArgument`); the range or possible outcomes of the random variable (represented

²This graph was generated using the OntoGraf plugin for Protégé, which comes with the default distribution of Protégé 4. The homepage of the OntoGraf project is <http://protegewiki.stanford.edu/wiki/OntoGraf>.

by the property `prowl2:hasPossibleValues`; and finally, its distribution (represented by the property `prowl2:hasProbabilityDistribution`).

In MEBN, every random variable has absurd as one of its possible values. Therefore, in PR-OWL this is also the case.

Figure A.3 presents the general hierarchy of the class `RandomVariable` and its children as well as the class `Absurd`. This is just part of the hierarchy shown in Figure A.1.

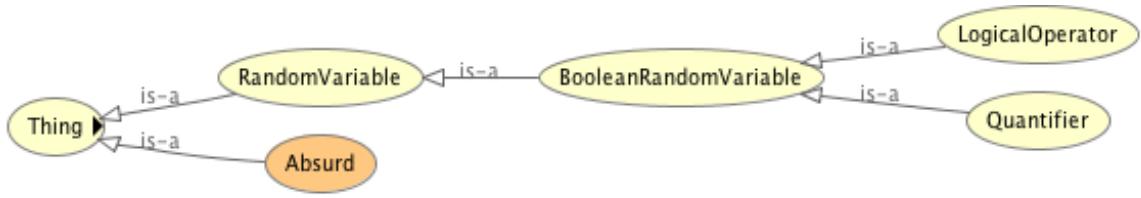


Figure A.3: The hierarchy of the `RandomVariable` class.

Figure A.4 presents not only the hierarchy but also the built-in random variables defined in PR-OWL. Although it is not explicitly shown, there is a “soft” link between `RandomVariable` and `Absurd`, represented by the restriction `hasPossibleValues` value ‘‘`http://www.pr-owl.org/pr-owl2.owl#Absurd`’’`^^anyURI`. This kind of “soft” link is a pattern that is used more than once in PR-OWL 2. It is used to express that an individual-instance is related to some class or data type by some property. The main reason was to keep within OWL DL and make use of OWL reasoners whenever possible. Otherwise, if PR-OWL 2 was defined using OWL Full, it would not be able to take advantage of the reasoners available for OWL, since they are usually not capable of reasoning with OWL Full ontologies.

For instance, in order to properly say that a random variable defines the uncertainty of some property, we would have to define the class `RandomVariable` having the restriction `definesUncertaintyOf` `some rdf:Property`. However, using such a restriction would require OWL Full, since according to [53], “IRIs from the reserved vocabulary other than

owl:Thing and owl:Nothing must not be used to identify classes in an OWL 2 DL ontology.” In order to keep PR-OWL 2 as an OWL 2 DL ontology, we avoided this restriction and replaced it by `definesUncertaintyOf` some `xsd:anyURI`. Note here that there are semantics that OWL-DL reasoners cannot capture, but PR-OWL reasoners are expected to respect.

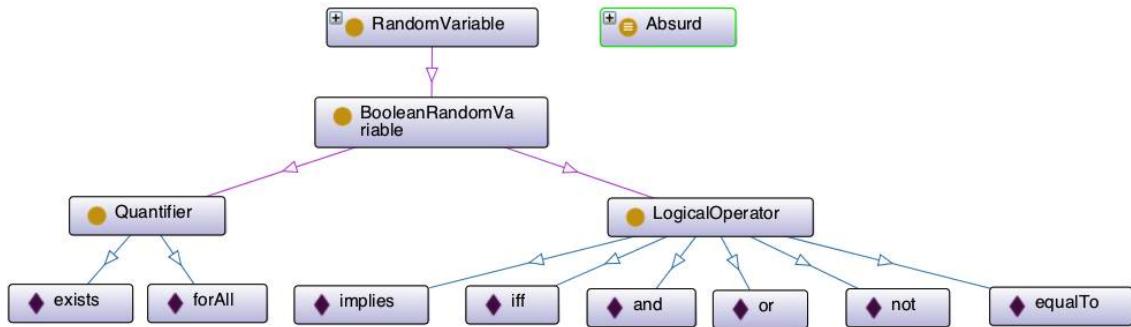


Figure A.4: Graph with the main concepts for defining random variables.

Although the possible values of a random variable is defined as any URI (by the restriction `hasPossibleValues only anyURI`), the semantics of PR-OWL defines that the only possible URI's are those that point to a class or a data type.

Listing A.1: Example of a random variable with the float data type as its possible value

```

1 Individual: RV.hasAnnualIncome
2   Types:
3     pr-owl2:RandomVariable
4   Facts:
5     pr-owl2:hasProbabilityDistribution RV.hasAnnualIncome.DD.dist1 ,
6     pr-owl2:hasArgument RV.hasAnnualIncome.MA.person ,
7     pr-owl2:hasPossibleValues "&xsd;float"^^xsd:anyURI ,
8     pr-owl2:definesUncertaintyOf "&ex;hasAnnualIncome"^^xsd:anyURI
  
```

An example of a data random variable is presented in Listing A.1, where the random variable `ex:RV.hasAnnualIncome` defines the uncertainty of the OWL property `ex:hasAnnualIncome`, its domain (argument) is `ex:RV.hasAnnualIncome.MA.person`, its

range is a `float` (for simplification sake, we assume the annual income is just a number), and its probability distribution is defined by `ex:RV.hasAnnualIncome.DD.dist1`.

More information on how to define arguments and how to define distributions are explained in sections A.3 and A.2, respectively. For now, it suffices to say that `ex:RV.hasAnnualIncome.MA.person` is an argument of type `ex:Person` and `ex:RV.hasAnnualIncome.DD.dist1` is a normal distribution with mean 50,000.00 and standard deviation 20,000.00.

An example of an object random variable is presented in Listing A.2, where the random variable `ex:RV.livesAt` defines the uncertainty of the OWL property `ex:livesAt`, its domain (argument) is `ex:RV.livesAt.MA.person`, its range is a `ex:Address`, and its probability distribution is defined by `ex:RV.livesAt.DD.dist1`.

Listing A.2: Example of a random variable with the class Address as its possible value

```

1 Individual: RV.livesAt
2   Types:
3     pr-owl2:RandomVariable
4   Facts:
5     pr-owl2:hasArgument  RV.livesAt.MA.person ,
6     pr-owl2:hasProbabilityDistribution  RV.livesAt.DD.dist1 ,
7     pr-owl2:definesUncertaintyOf  "&ex;livesAt"^^xsd:anyURI ,
8     pr-owl2:hasPossibleValues  "&ex;Address"^^xsd:anyURI

```

More information on how to define arguments and how to define distributions are explained in sections A.3 and A.2, respectively. For now, it suffices to say that `ex:RV.livesAt.MA.person` is an argument of type `ex:Person` and `ex:RV.livesAt.DD.dist1` is uniform distribution over all the individuals of class `ex:Address`.

`BooleanRandomVariable` is a special kind of `RandomVariable`, which represents random variables that can only have Boolean values as their possible values or range, guaranteed by the restriction `hasPossibleValues` value '`'http://www.w3.org/2001/XMLSchema#boolean'`'`^^anyURI` (see Figure A.5).

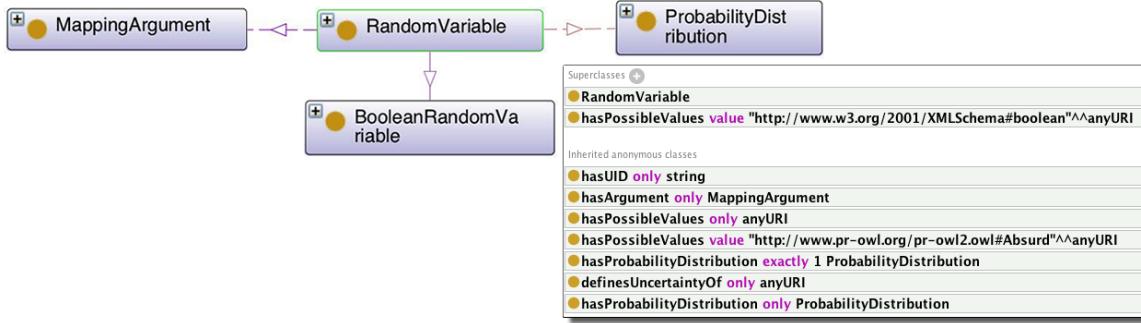


Figure A.5: The OWL restrictions of the `BooleanRandomVariable` class.

An example of a Boolean random variable is presented in Listing A.3, where the random variable `ex:RV.isRelated` defines the uncertainty of the OWL property `ex:isRelated`, its domain (arguments) are `ex:RV.isRelated.MA.person1` and `ex:RV.isRelated.MA.person2`, its range is Boolean, and its probability distribution is defined by `ex:RV.isRelated.PT.dist1`.

Listing A.3: Example of a Boolean random variable

```

1 Individual: RV.isRelated
2   Types:
3     pr-owl2:BooleanRandomVariable
4   Facts:
5     pr-owl2:hasArgument  RV.isRelated.MA.person1 ,
6     pr-owl2:hasArgument  RV.isRelated.MA.person2 ,
7     pr-owl2:hasProbabilityDistribution  RV.isRelated.PT.dist1 ,
8     pr-owl2:definesUncertaintyOf  "&ex;isRelated"^^xsd:anyURI

```

More information on how to define arguments and how to define distributions are explained in sections A.3 and A.2, respectively. For now, it suffices to say that `ex:RV.isRelated.MA.person1` and `ex:RV.isRelated.MA.person2` are both arguments of type `ex:Person` and `ex:RV.isRelated.PT.dist1` has a distribution defined by a PR-OWL table.

`LogicalOperator` is a special kind of `BooleanRandomVariable`, which represents

first-order logic (FOL) operators. These logic operators random variables are mostly used to express FOL formulas using MEBN expressions (see Section A.3 for details on how to define FOL formulas). Figure A.6 presents the OWL restrictions for this class.

Since these operators can represent more expressive formulas than those represented in OWL DL, there is no explicit mapping of these RVs to OWL properties. This is a special case for these built-in random variables. Nevertheless, they can be mapped (in a future version) to an OWL ontology that defines FOL logic operators.

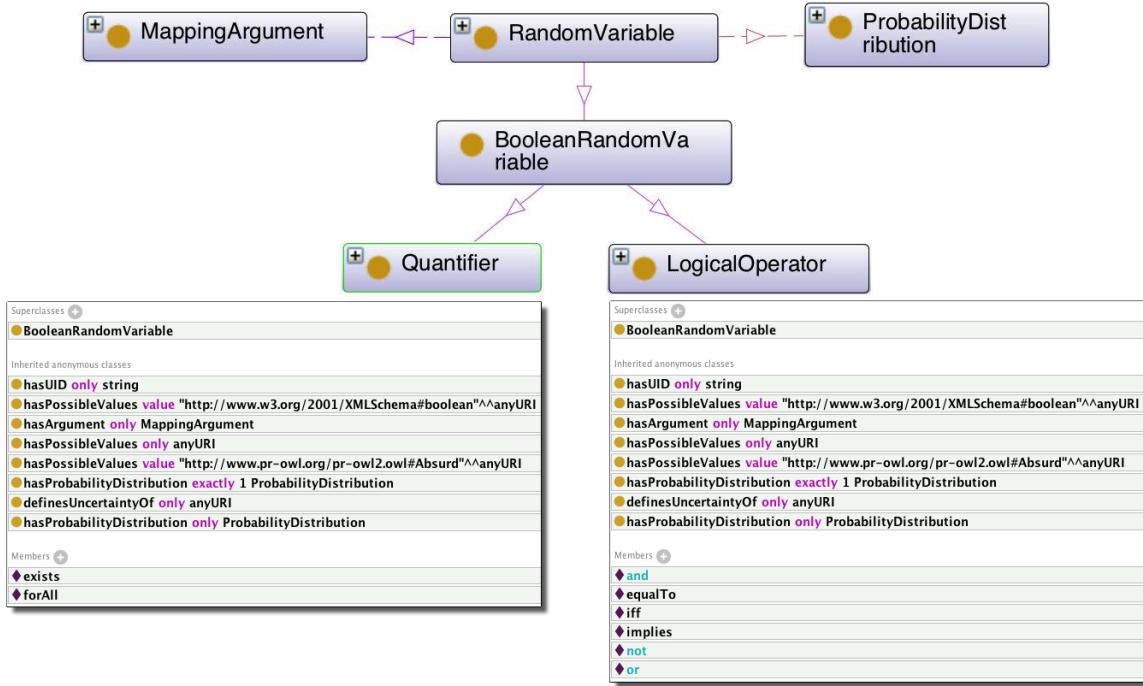


Figure A.6: The OWL restrictions of the classes `LogicalOperator` and `Quantifier`.

The built-in logical operators available in PR-OWL 2 are the same and also represented as instances as in PR-OWL 1. Namely they are:

`and` represents the FOL 'and' operator and must have two arguments.

`or` represents the FOL 'or' operator and must have two arguments.

`not` represents the FOL 'not' operator and must have one argument.

`equalTo` represents the FOL '=' operator and must have two arguments.

`implies` represents the FOL ' \Rightarrow ' operator, which is an 'if then' statement, and must have two arguments.

`iff` represents the FOL ' \Leftrightarrow ' operator, which is an 'if and only if' statement, and must have two arguments.

`Quantifier` is a special kind of `BooleanRandomVariable`, which represents first-order (FOL) quantifiers. These quantifier RVs are mostly used to express FOL formulas using MEBN expressions (see Section A.3 for details on how to define FOL formulas). Figure A.6 presents the OWL restrictions for this class.

Since these quantifiers can represent more expressive formulas than those represented in OWL DL, there is no explicit mapping of these RVs to OWL properties. This is a special case for these built-in random variables. Nevertheless, they can be mapped (in a future version) to an OWL ontology that defines FOL quantifiers.

The built-in quantifiers available in PR-OWL 2 are the same and also represented as instances as in PR-OWL 1. Namely they are:

`exists` represents the FOL ' \exists ' quantifier and must have two arguments (one exemplar and one formula which it is quantifying over). If more than one variable needs to be quantified over, *i.e.*, if you need more than one exemplar on the same formula, it is necessary to nest this quantifier with another quantifier over the other bound variable. This restriction is to be consistent with the MEBN specification presented in [78].

`forAll` represents the FOL ' \forall ' quantifier and must have two arguments (one exemplar and one formula which it is quantifying over). If more than one variable needs to be quantified over, *i.e.*, if you need more than one exemplar

on the same formula, it is necessary to nest this quantifier with another quantifier over the other bound variable. This restriction is to be consistent with the MEBN specification presented in [78].

A.2 MEBN Main Elements

Figure A.7 presents the general hierarchy of the classes that represent the main elements of the MEBN logic. This is just part of the hierarchy shown in Figure A.1.

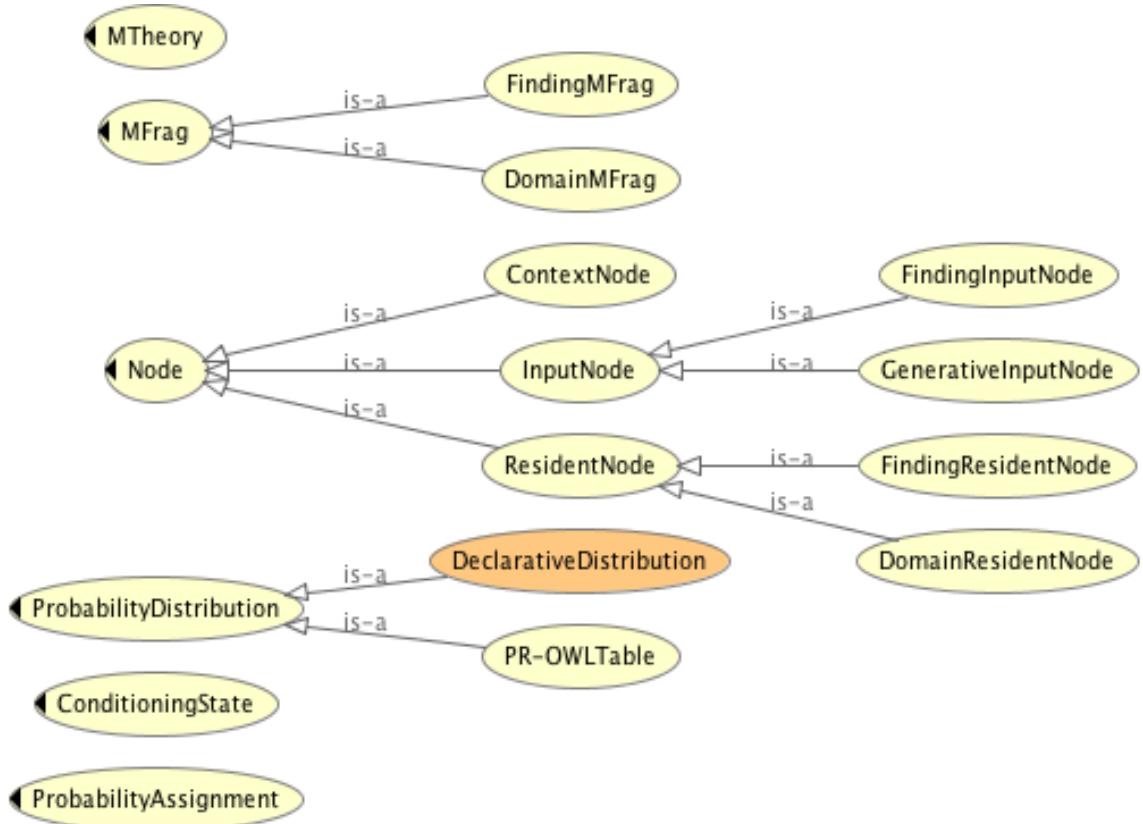


Figure A.7: The hierarchy of the main classes that represent MEBN elements.

MTheory is a collection of MEBN Fragments (MFrags) that satisfies consistency constraints ensuring the existence of a joint distribution over the random variables mentioned in

the MEBN Theory. Incomplete MTheories admit multiple different distributions. In PR-OWL, the class MTheory allows a probabilistic ontology to have more than one valid MTheory to represent its RVs. In addition, one MFragment can be part of more than one MTheory. Figure A.8 presents the OWL restrictions for this class.

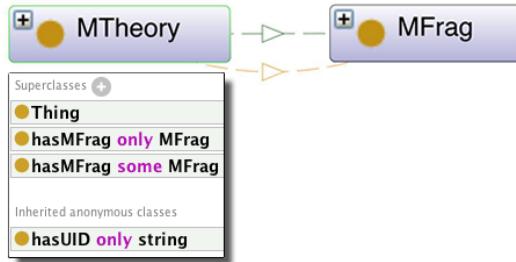


Figure A.8: The OWL restrictions of the MTheory class.

Listing A.4 presents a simple MTheory called `FraudIdentificationInPublicProcurement`, which is a collection of two MFrags, `PersonalInformation` and `Finding1`.

Listing A.4: MTheory example

```

1 Individual: MTheory.FraudIdentificationInPublicProcurement
2   Types:
3     pr-owl2:MTheory
4   Facts:
5     pr-owl2:hasMFragment MFragment.Finding1 ,
6     pr-owl2:hasMFragment MFragment.PersonalInformation

```

Figure A.9 presents the main MEBN elements and their relations. An MEBN theory is composed of one or more MEBN fragments. An MEBN fragment is composed of one or more nodes and it can also have ordinary variables and exemplars, which are used in first-order logic formulas or terms (represented by MEBN expressions). A node has its representation defined by a MEBN expression. An MEBN expression is a first-order logic formula or term, of a specific type, represented by the random variable (*e.g.*, an equal formula has the `equalTo` random variable as its type). A

random variable has a default probability distribution. Finally, there are three types of nodes. A resident node can be parent of another resident node and has a probability distributions associated with it. An input node represents a first-order formula or term that influences the distribution of at least one resident node, but has its distribution defined by the random variables it uses (not defined within that MEBN fragment). Finally, context nodes are first-order formulas that have to be true in order for the probability distributions defined within their MEBN fragment to be valid.

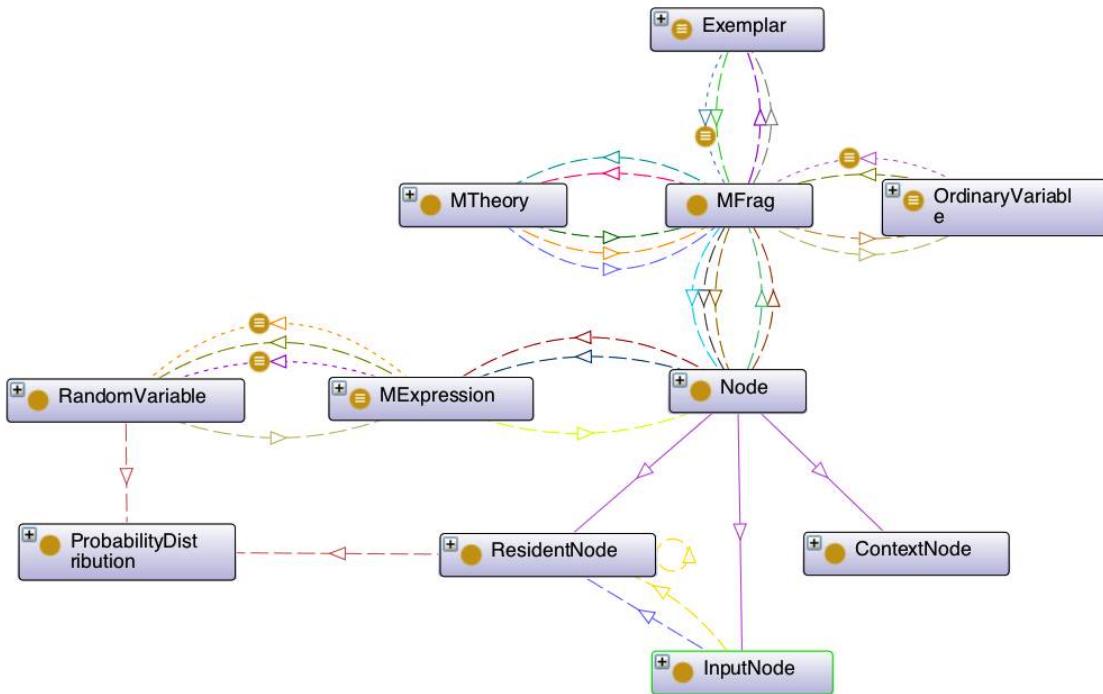


Figure A.9: Graph of the main MEBN elements and their relations.

MFrag is the basic structure of any MEBN logic model. MEBN Fragments (MFrgs) represent influences among clusters of related random variables and can portray repeated patterns. To represent patterns, MEBN uses ordinary variables, which are free variables in formulas or terms that can be substituted by entity identifiers, and/or exemplars,

which correspond to bound variables in quantified formulas. In PR-OWL, each subclass of the `MFrag` class represents an MEBN fragment.

Every `MFrag` has at least one node (at least one resident node). However, it can also have input nodes, ordinary variables and exemplars, and be part of one or more MEBN theories. Figure A.10 presents the OWL restrictions for this class.

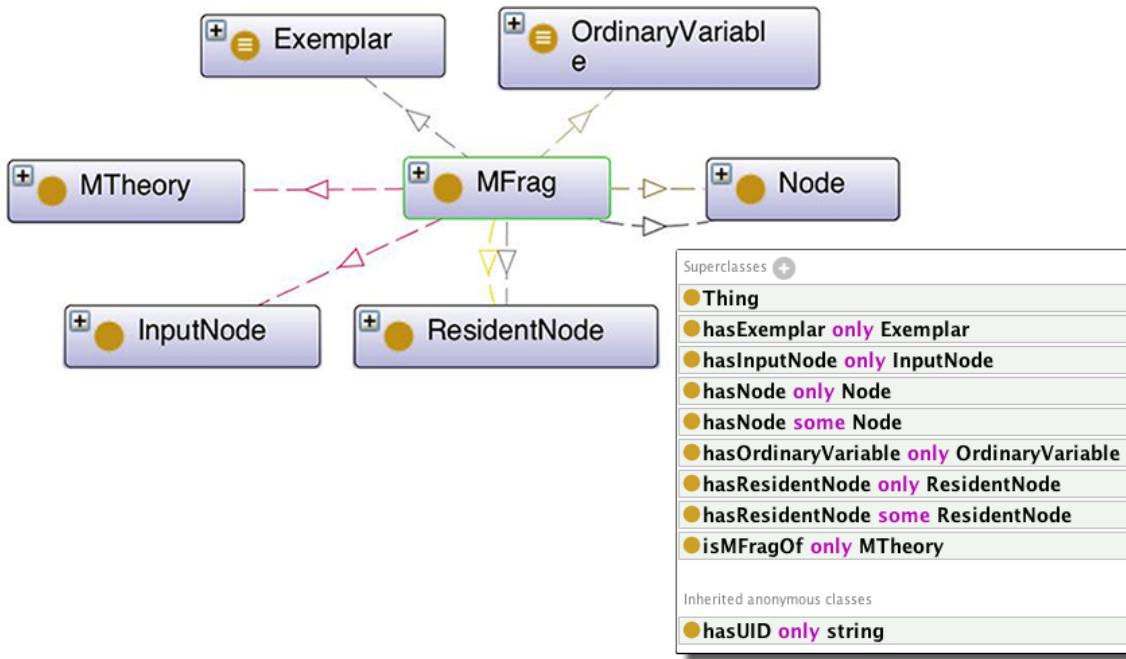


Figure A.10: The OWL restrictions of the `MFrag` class.

An `MFrag` is either a domain `MFrag` or a finding `MFrag`, therefore examples of `MFrag`s will be showed after each type is described in more detail.

`DomainMFrag` is the subclass of class `MFrag` that includes all the domain-specific `MFrag`s. It is disjoint from the class `FindingMFrag`. All generative `MFrag`s created by the ontology engineer (*i.e.* the domain expert) are individuals of this class.

A domain `MFrag` differs from an `MFrag` by allowing the use of context nodes,

which define the necessary conditions for the relations and distributions defined on the MFragment to be valid. Moreover, it only accepts resident nodes of type `DomainResidentNode` and input nodes of type `GenerativeInputNode`. Figure A.11 presents the OWL restrictions for this class.

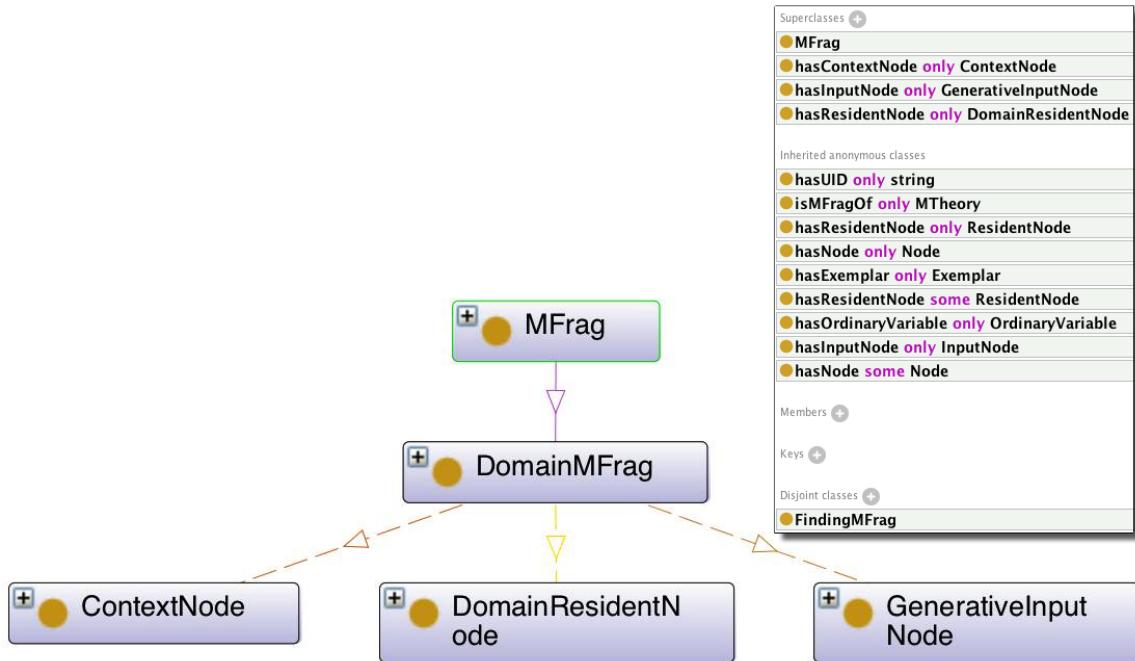


Figure A.11: The OWL restrictions of the `DomainMFragment` class.

Figure A.12 presents the main concepts in defining a domain-specific MFragment and its relations. A domain-specific MFragment, besides exemplars and ordinary variables, has more specific input nodes and resident nodes (namely, `GenerativeInputNode` and `DomainResidentNode`, respectively). A domain resident node only accepts simple MEBN expressions, while generative input nodes accept any type of MEBN expression. Besides input nodes and resident nodes, a domain MFragment has context nodes, which are responsible for defining the restrictions that must be satisfied in order for the probabilistic relations and distributions within that MFragment to be valid. As a context node represents a constraint that must be true,

it only accepts Boolean MEBN expressions.

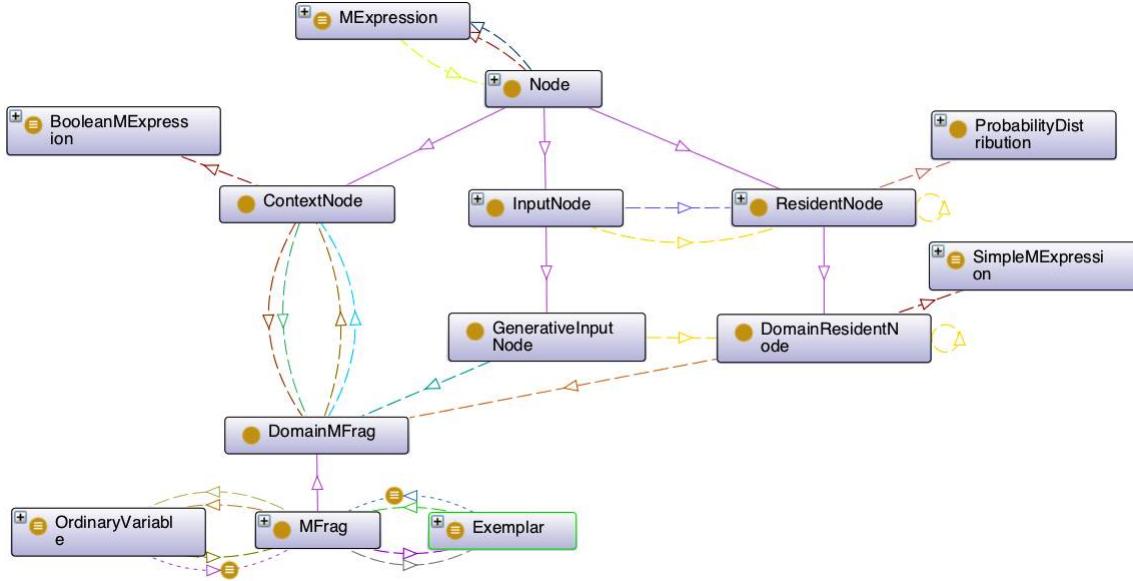


Figure A.12: Graph with the main elements necessary to define a domain-specific MEBN fragment.

Listing A.5 presents an MFragment called `ex:PersonalInformation`, that defines the distribution of two different people being related given they live at the same address.

Listing A.5: Domain MFragment example

```

1 Individual: MFragment. PersonalInformation
2   Types:
3     pr-owl2:DomainMFragment
4   Facts:
5     pr-owl2:isMFragmentOf MTheory. FraudIdentificationInPublicProcurement ,
6     pr-owl2:hasOrdinaryVariable MFragment. PersonalInformation .OV. person1 ,
7     pr-owl2:hasOrdinaryVariable MFragment. PersonalInformation .OV. person2 ,
8     pr-owl2:hasContextNode MFragment. PersonalInformation .CN. not1 ,
9     pr-owl2:hasInputNode MFragment. PersonalInformation .GIN. equalTo1 ,
10    pr-owl2:hasResidentNode MFragment. PersonalInformation .DRN. isRelated
  
```

`FindingMFragment` is used to convey information about findings, which is the default way

of entering evidence in an MEBN theory so a probabilistic algorithm can be applied to perform inferences regarding the new evidence. It has no context nodes, only one input and one resident node. Figure A.13 presents the OWL restrictions for this class.

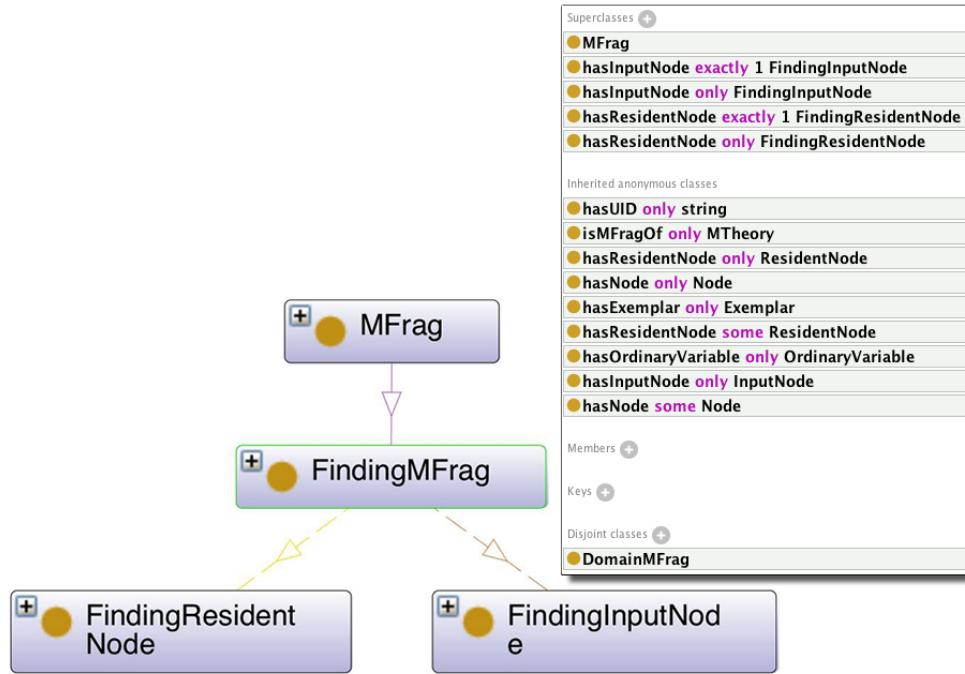


Figure A.13: The OWL restrictions of the **FindingMfrag** class.

Figure A.14 presents the main concepts involved in entering findings in an MEBN theory. Besides having ordinary variables and exemplars that can be used in formulas, which defines the finding, a finding MFrag only has one finding resident node and one finding input node. Since the idea of a finding is to say that some formula is true, the only MEBN expression allowed for finding resident nodes and input nodes is Boolean.

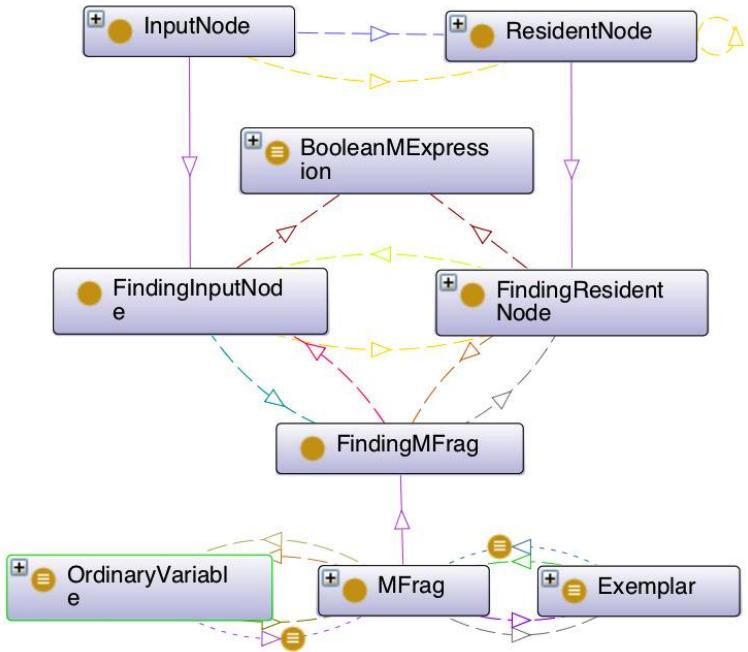


Figure A.14: Graph with the main concepts and their relations for defining findings in an MEBN theory.

The MEBN representation of a finding is shown in Figure A.15. It can be seen that both the finding resident and input nodes represent the same Boolean expression. The difference is that a finding resident node only has true and absurd as possible values and its local probability distribution is defined as true if the input node is true and absurd otherwise. The finding input node has the same behavior as any regular input node, except that it can only represent Boolean expressions.

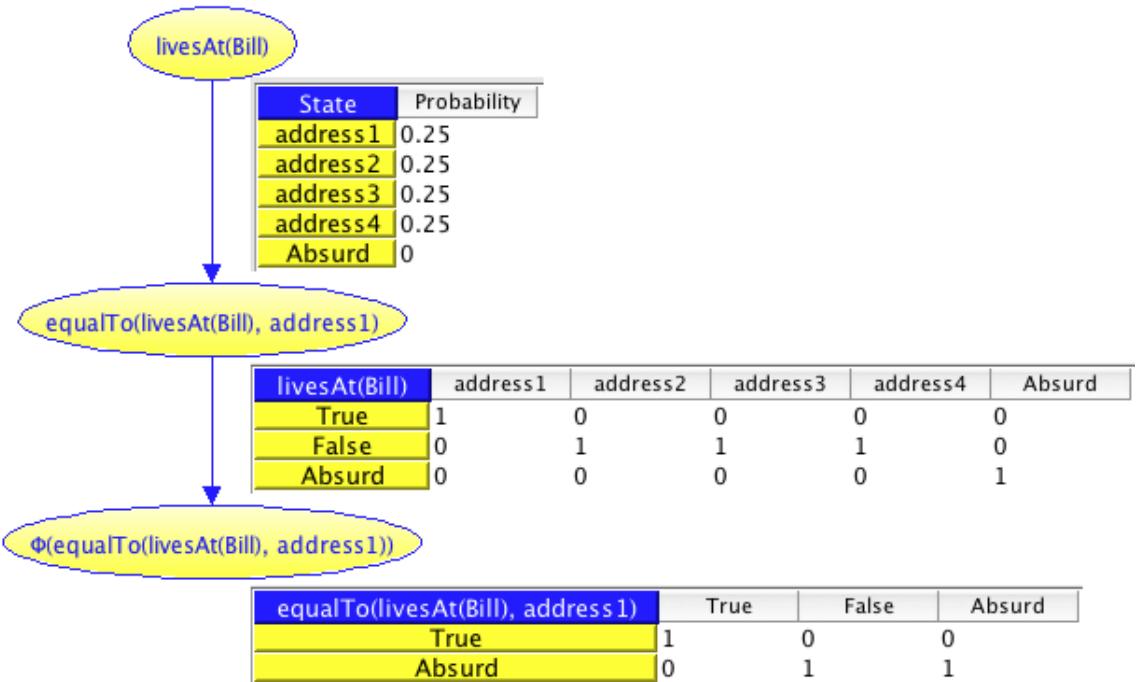


Figure A.15: Bayesian network showing the pattern for representing findings in MEBN.

Listing A.6 presents a finding MFragment called `Finding1`, that states that `equalTo(hasAnnualIncome(Bill), 75,000.00)`, represented by the finding resident node `ex:MFragment.FRN.equalTo1`.

Listing A.6: Finding MFragment example

```

1 Individual: MFragment.Finding1
2   Types:
3     pr-owl2:FindingMFragment
4   Facts:
5     pr-owl2:isMFragmentOf MTheory.FraudIdentificationInPublicProcurement ,
6     pr-owl2:hasInputNode MFragment.Finding1.FIN.equalTo1 ,
7     pr-owl2:hasResidentNode MFragment.Finding1.FRN.equalTo1

```

`Node` is part of an MFragment and it can define the distribution of a random variable within that MFragment (a resident node, represented by the class `ResidentNode`), a random variable that influence the distribution of nodes within that MFragment but has its distribution

defined somewhere else (an input node, represented by the class `InputNode`), or a random variable that expresses the context in which the probability distributions within that MFragment are valid (a context node, represented by the class `ContextNode`). `ResidentNode`, `InputNode`, and `ContextNode` are disjoint classes.

In all cases, the random variable represented by the node is defined as a MEBN expression (for more information about `MExpression` see Section A.3). Besides that, every node is resident in exactly one MFragment. Figure A.16 presents the OWL restrictions for this class.

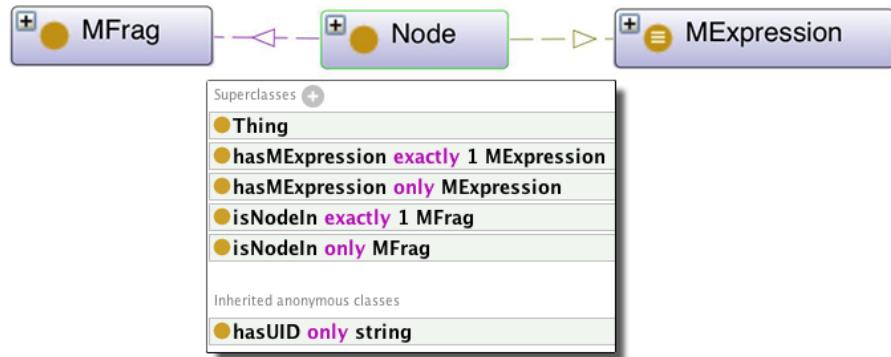


Figure A.16: The OWL restrictions of the `Node` class.

Example of nodes will be given for the more specific types of nodes (`DomainResidentNode`, `FindingResidentNode`, `ContextNode`, `FindingInputNode`, and `GenerativeInputNode`).

`ResidentNode` is associated with a random variable and has its probability distribution defined within that MFragment. Besides its distribution, a `ResidentNode` can have parents (either `InputNode` or another `ResidentNode`) and it is a resident node in exactly one MFragment. `ResidentNode` is disjoint from `InputNode` and `ContextNode`. Figure A.17 presents the OWL restrictions for this class.

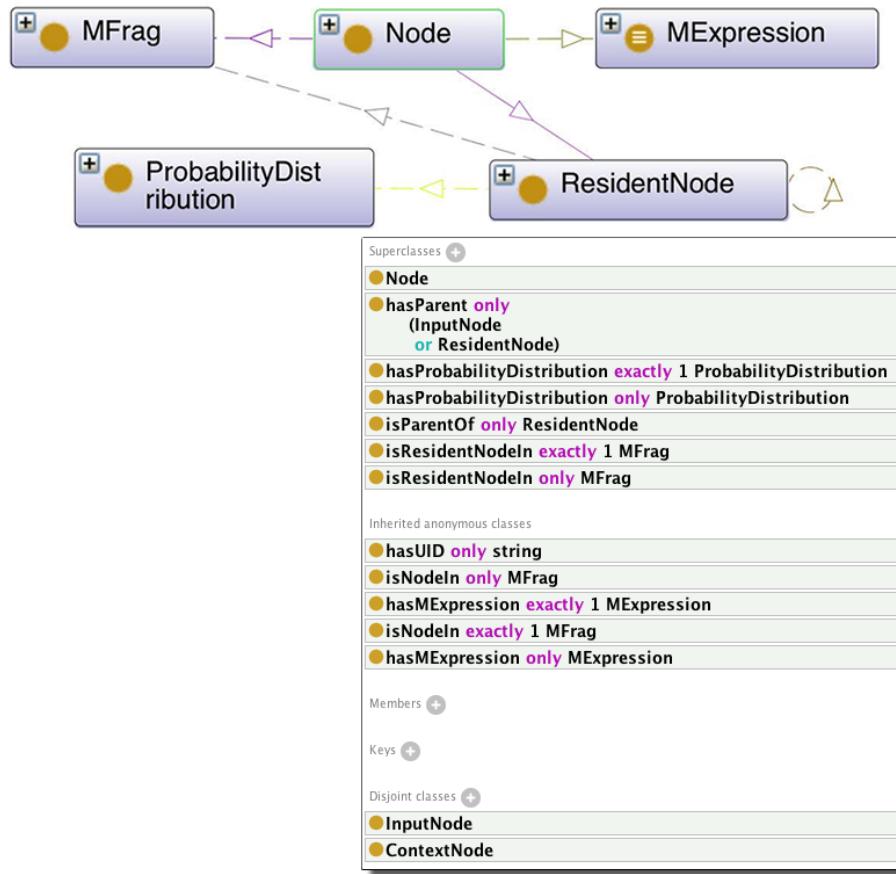


Figure A.17: The OWL restrictions of the `ResidentNode` class.

Example of resident nodes will be given for the more specific types of resident nodes (`DomainResidentNode` and `FindingResidentNode`).

`DomainResidentNode` is the subclass of `ResidentNode` that includes all domain-specific resident nodes. It is disjoint from `FindingResidentNode`.

A `DomainResidentNode` only defines a distribution over a simple MEBN expression (see `SimpleMExpression` in Section A.3 for more information). It can only have parents of type `GenerativeInputNode` or `DomainResidentNode`. Finally, a `DomainResidentNode` can only be defined in exactly one `DomainMFrag`. Figure A.18 presents the OWL restrictions for this class.

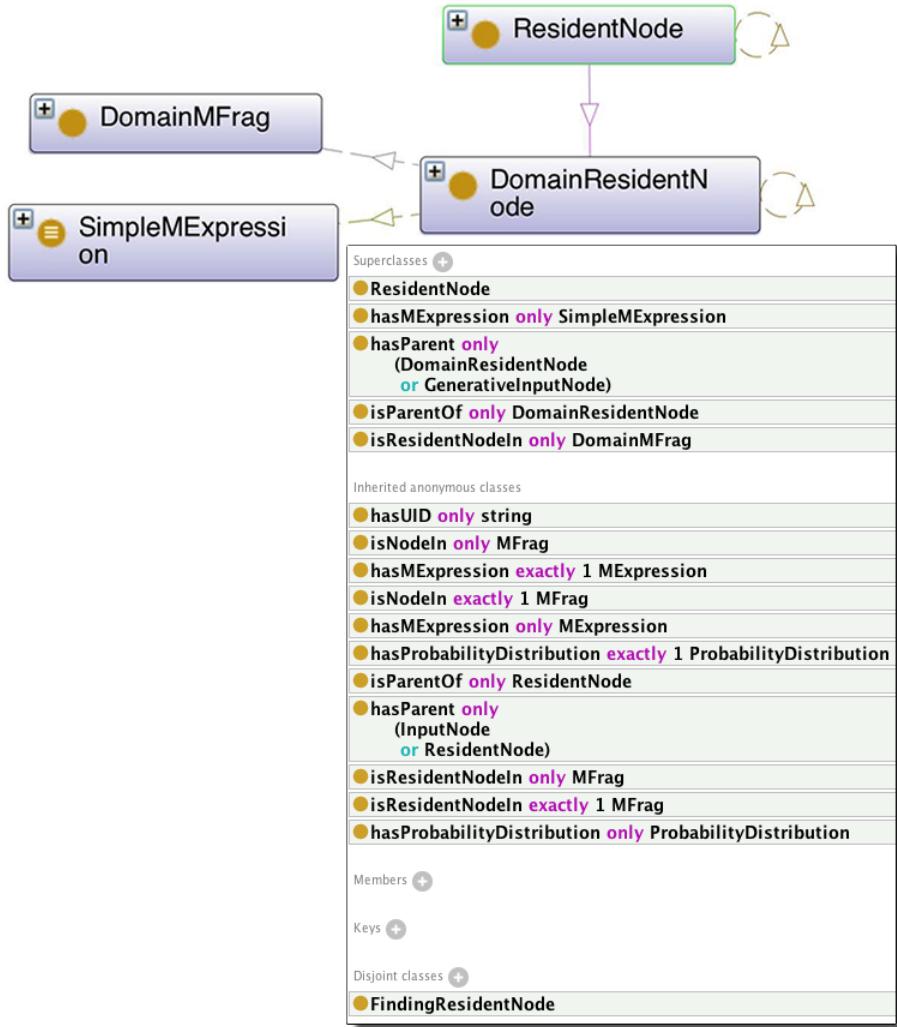


Figure A.18: The OWL restrictions of the `DomainResidentNode` class.

Listing A.7 presents a domain resident node with a local probability distribution for the random variable `isRelated(person1, person2)` (see `SimpleMExpression` in Section A.3 for more details on how to define such MEBN expression for this resident node), conditioned on the input node `equalTo(livesAt(person1), livesAt(person2))` (see `GenerativeInputNode` later in this section for more details about this input node). Although the distribution is not shown in the example (see PR-OWLTable

later in this Section), it basically says the if two people live at the same address they are more likely to be related.

Listing A.7: Domain resident node example

```

1 Individual: MFragment.PersonalInformation.DRN.isRelated
2   Types:
3     pr-owl2:DomainResidentNode
4   Facts:
5     pr-owl2:isResidentNodeIn MFragment.PersonalInformation ,
6     pr-owl2:hasParent MFragment.PersonalInformation.GIN.equalTo1 ,
7     pr-owl2:hasMExpression
8       MFragment.PersonalInformation.DRN.isRelated.SME.isRelated ,
9     pr-owl2:hasProbabilityDistribution
10      MFragment.PersonalInformation.DRN.isRelated.PT.dist1

```

`FindingResidentNode` is the subclass of `ResidentNode` that includes all finding nodes. Finding nodes convey new evidence into a probabilistic system via a `FindingMFragment`. It is disjoint from `DomainResidentNode`.

A `FindingResidentNode` can only represent a Boolean MEBN expressions (see `BooleanMExpression` in Section A.3 for more information), since it is being stated that this Boolean expression is true (a finding). It has only one parent and it has to be of the type `FindingInputNode`. It cannot be parent of any other node. Finally, it can only be defined in exactly one `FindingMFragment`. Figure A.19 presents the OWL restrictions for this class.

In order to understand how to use this node, see `FindingMFragment` in this section, especially the explanation of Figures A.14 and A.15.

Listing A.8 presents a finding resident node for the Boolean expression `equalTo(hasAnnualIncome(Bill), 75,000.00)` (see `BooleanMExpression` in Section A.3 for more details on how to define such MEBN expression for this finding resident node), conditioned on the input node `equalTo(hasAnnualIncome(Bill), 75,000.00)` (see `FindingInputNode` later in this section for more details about this input node). Again, to understand how to define findings see `FindingMFragment` in this Section.

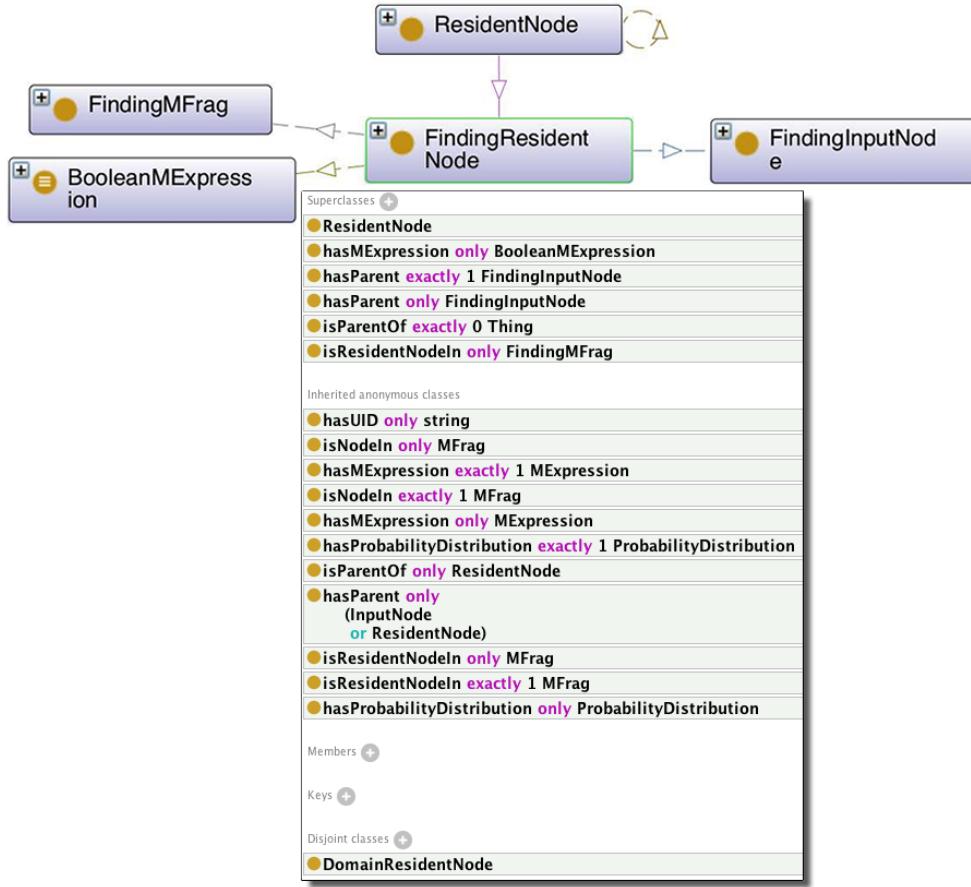


Figure A.19: The OWL restrictions of the `FindingResidentNode` class.

Listing A.8: Finding resident node example

```

1 Individual: Mfrag.Finding1.FRN.equalTo1
2 Types:
3   pr-owl2:FindingResidentNode
4 Facts:
5   pr-owl2:hasParent Mfrag.Finding1.FIN.equalTo1 ,
6   pr-owl2:hasMExpression Mfrag.Finding1.FRN.equalTo1.BME.equalTo1

```

`ContextNode` defines a constraint on the `Mfrag` where it is defined. Often these constraints define the type of arguments used for resident nodes (*e.g.*, `isA(mother, Person)`, `isA(person, Person)`), reference constraints between these arguments

(e.g., `mother = motherOf(person)`), etc. It is disjoint from `ResidentNode` and `InputNode`.

A `ContextNode` can only represent a Boolean MEBN expression (see `BooleanMExpression` in Section A.3 for more information), since it is being stated that this Boolean expression, the constraint, has to be true (valid). Finally, it can only be a context node in exactly one `DomainMFragment`. Figure A.20 presents the OWL restrictions for this class.

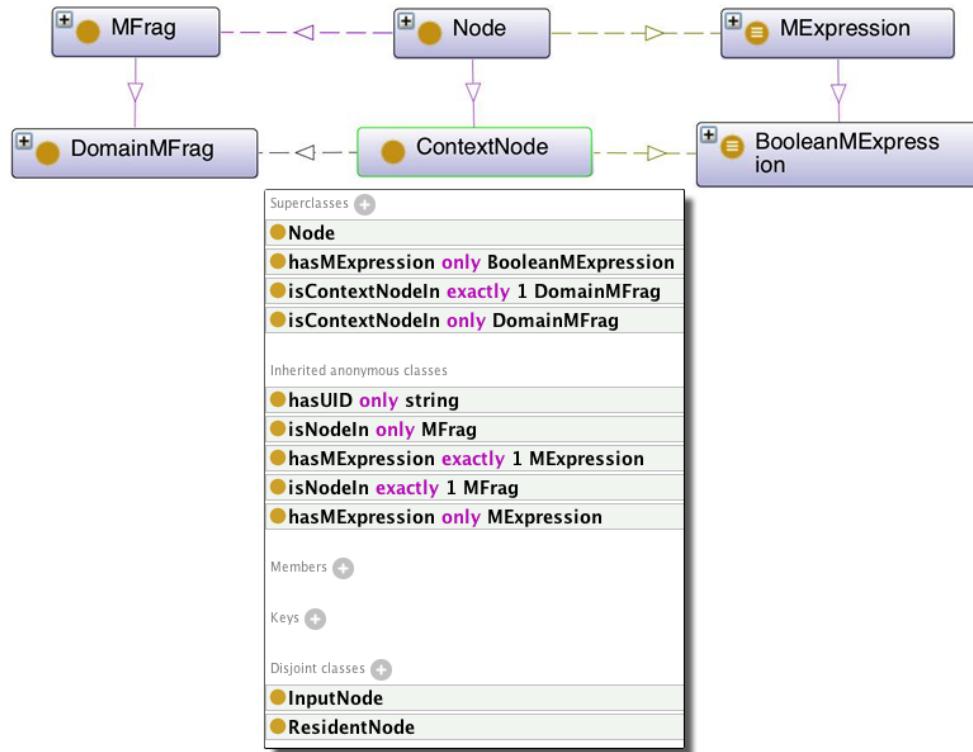


Figure A.20: The OWL restrictions of the `ContextNode` class.

Listing A.9 presents a context node for the Boolean expression `not(equalTo(person1, person2))` (see `BooleanMExpression` in Section A.3 for more details on how to define such MEBN expression). In other words, this context node states that within `MFragment ex:MFragment.PersonalInformation`,

`person1` and `person2` have to be different.

Listing A.9: Context node example

```

1 Individual: MFragment.PersonalInformation.CN.not1
2   Types:
3     pr-owl2:ContextNode
4   Facts:
5     pr-owl2:isContextNodeIn MFragment.PersonalInformation ,
6     pr-owl2:hasMExpression MFragment.PersonalInformation.CN.not1.BME.not1

```

`InputNode` is a random variable (RV) that has its distribution defined somewhere else, but its value influence some `ResidentNode` within that `MFragment`. Therefore, it has to be a parent of some `ResidentNode` and it can only be parent of resident nodes. Finally, it can only be an input node in exactly one `DomainMFragment`. It is disjoint from `ResidentNode` and `ContextNode`. Figure A.21 presents the OWL restrictions for this class.

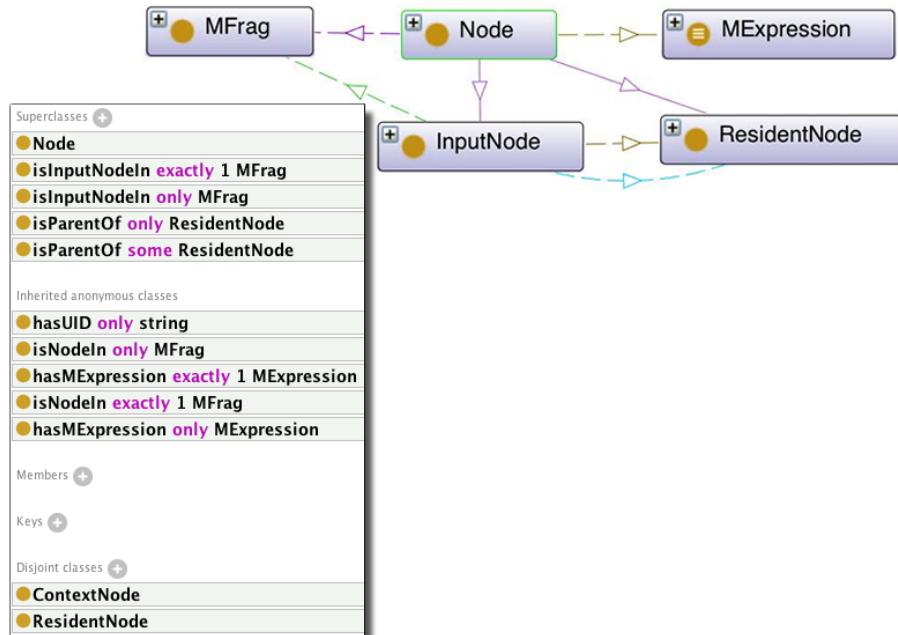


Figure A.21: The OWL restrictions of the `InputNode` class.

Example of input nodes will be given for the more specific types of input nodes (`FindingInputNode` and `GenerativeInputNode`).

`FindingInputNode` represents a Boolean MEBN expression (see `BooleanMExpression` in Section A.3 for more information), which influences some `FindingResidentNode` within that MFrag. In fact, it can only be parent of one node, and this node has to be of type `FindingResidentNode`. It can only be an input node in exactly one `FindingMFrag`. Finally, it is disjoint from `GenerativeInputNode`. Figure A.22 presents the OWL restrictions for this class.

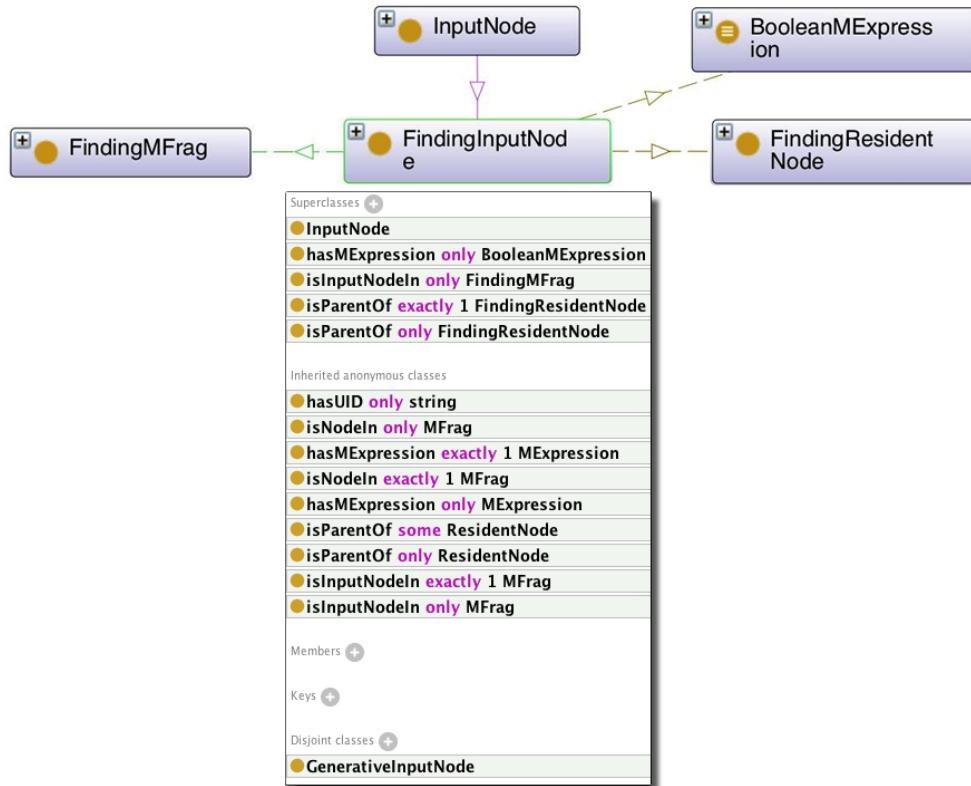


Figure A.22: The OWL restrictions of the `FindingInputNode` class.

In order to understand how to use this node, see `FindingMFrag` in this

section, especially the explanation of Figures A.14 and A.15.

Listing A.10 presents a finding input node for the Boolean expression `equalTo(hasAnnualIncome(Bill), 75,000.00)` (see `BooleanMExpression` in Section A.3 for more details on how to define such MEBN expression for this finding input node). It is parent of the finding resident node `equalTo(hasAnnualIncome(Bill), 75,000.00)` (see `FindingResidentNode` in this section for more details about this resident node). Again, to understand how to define findings see `FindingMFragment` in this Section.

Listing A.10: Finding input node example

```
1 Individual: MFrag.Finding1.FIN.equalTo1
2   Types:
3     pr-owl2:FindingInputNode
4   Facts:
5     pr-owl2:isParentOf MFrag.Finding1.FR.N.equalTo1 ,
6     pr-owl2:hasMExpression MFrag.Finding1.FIN.equalTo1.BME.equalTo1
```

`GenerativeInputNode` is a random variable (RV) that has its distribution defined somewhere else, but its value influences some `DomainResidentNode` within that `MFragment`. It can only be an input node in exactly one `DomainMFragment`. Finally, it is disjoint from `FindingInputNode`. Figure A.23 presents the OWL restrictions for this class.

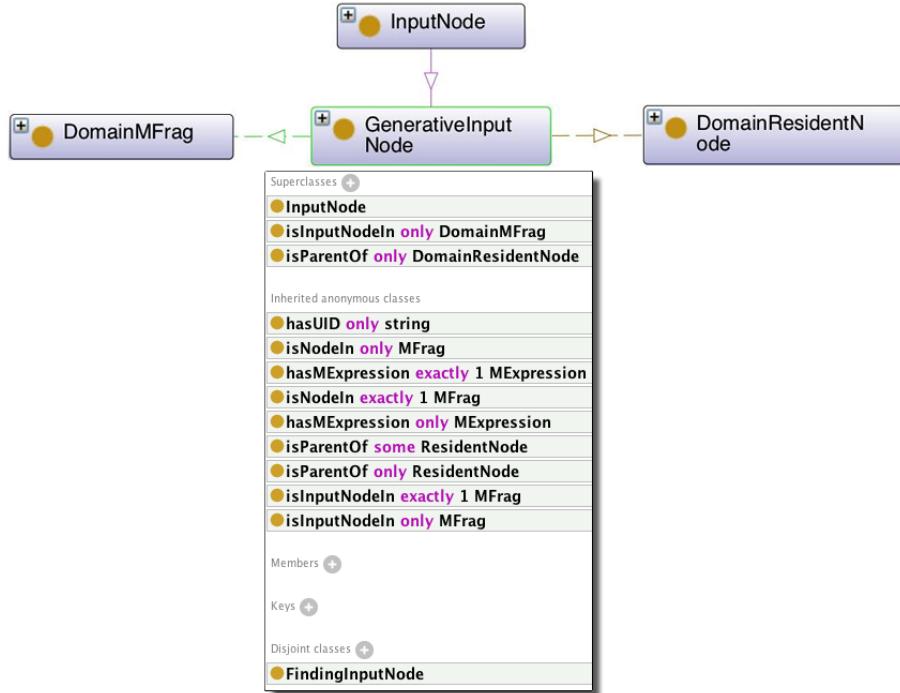


Figure A.23: The OWL restrictions of the `GenerativeInputNode` class.

Listing A.11 presents a generative input node for the MEBN expression `equalTo(livesAt(person1), livesAt(person2))` defined for the Mfrag `Mfrag.PersonalInformation`. It is parent of the resident node `isRelated(person1, person2)` (see `DomainResidentNode` in this Section for more details about this resident node). The idea is basically that if two people live at the same address they are more likely to be related.

Listing A.11: Generative input node example

```

1 Individual: Mfrag. PersonalInformation .GIN. equalTo1
2   Types:
3     pr-owl2:GenerativeInputNode
4   Facts:
5     pr-owl2:isInputNodeIn  Mfrag. PersonalInformation ,
6     pr-owl2:isParentOf    Mfrag. PersonalInformation .DRN. isRelated
7     pr-owl2:hasMExpression  Mfrag. PersonalInformation .GIN. equalTo1 .BME.
          equalTo1

```

`ProbabilityDistribution` is used to define the local distributions for each resident node (these local distributions apply only if all context nodes in the Mfrag are satisfied), and the default distribution for a random variable (to be used if none of the local distributions in any of its home Mfrags applies). A probability distribution can be described using an application dependent declarative format, such as UnBBayes local probability distribution (LPD), or via a PR-OWL table (which has probability assignments as its cells). Figure A.24 presents the OWL restrictions for this class.

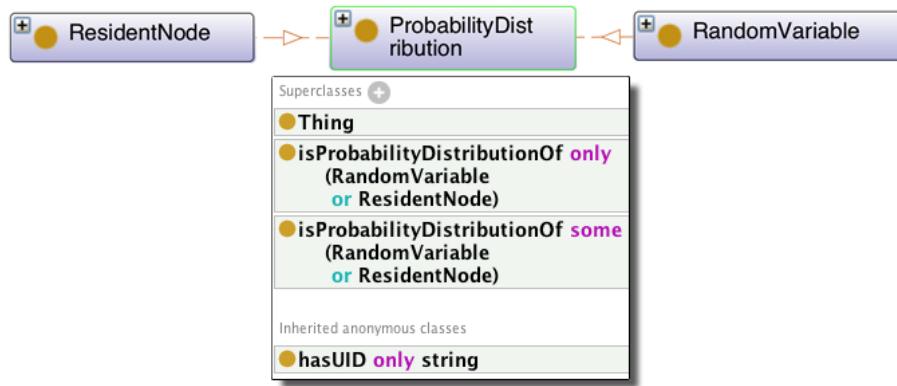


Figure A.24: The OWL restrictions of the `ProbabilityDistribution` class.

Figure A.25 presents the main concepts and their relations for defining probability distributions. It can be seen that both `ResidentNode` and `RandomVariable` have probability distributions. There are two types of probability distributions, `PR-OWLTable` and `DeclarativeDistribution`. A `PR-OWLTable` has probability assignments (`ProbabilityAssignment`), which depends on conditioning states from the parents (`ConditioningState`). A `DeclarativeDistribution` is defined by some script, which follows some application-specific grammar.

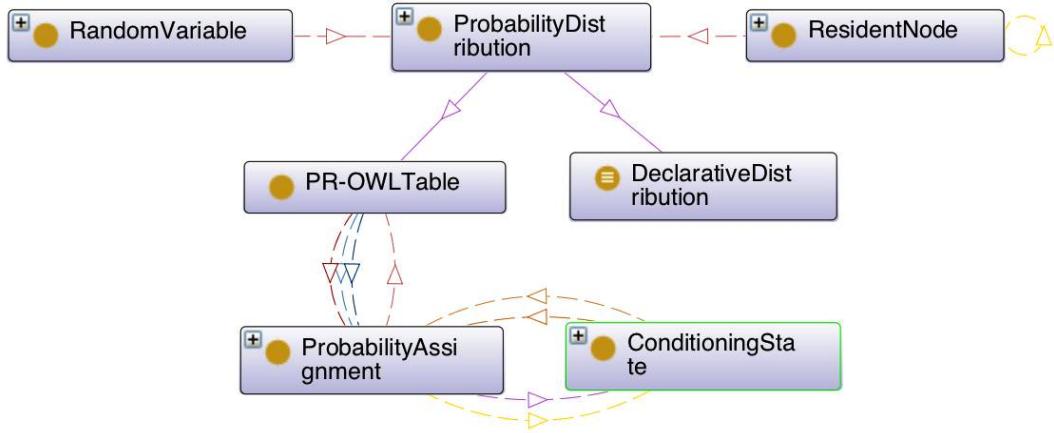


Figure A.25: Graph with the main concepts and their relations for defining probabilistic distributions.

`DeclarativeDistribution` is a distribution that is conveyed via a `xsd:string` data type, using a specific format defined in the `hasDeclaration` data type property. In order to allow an MEBN algorithm to work, a parser should be able to retrieve the probability distribution information in the format it is stored and then pass that information to the MEBN algorithm in its own application-specific format.

Figure A.26 presents the OWL restrictions for this class.

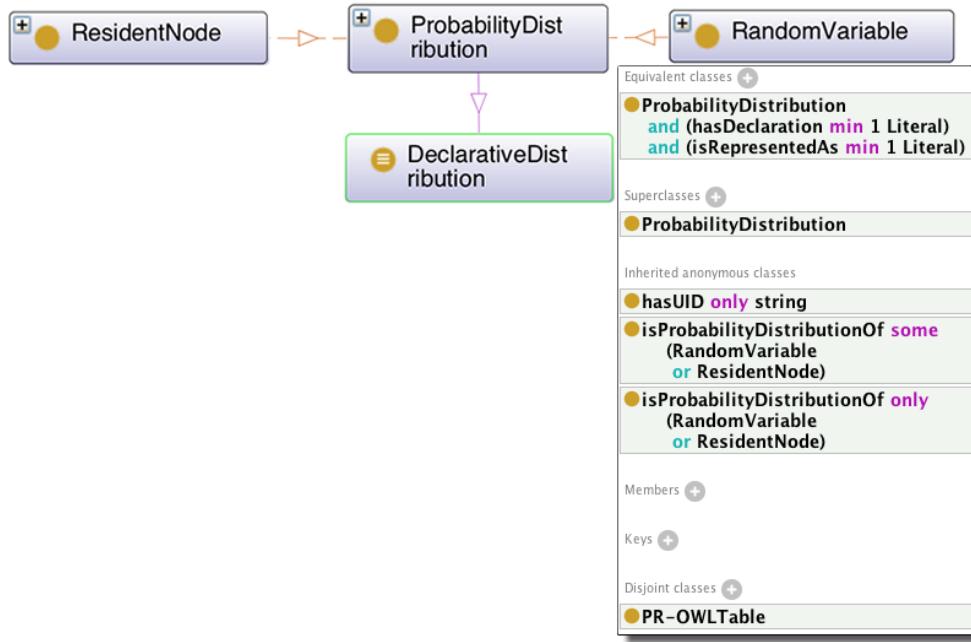


Figure A.26: The OWL restrictions of the `DeclarativeDistribution` class.

Describing a declarative probability distribution is a much more compact and flexible way of conveying the distribution. However, it assumes that a PR-OWL reasoner would understand the format in which the information is stored. PR-OWL tables, on the other hand, convey probability distributions in a more interoperable way, but are not flexible enough to represent complex distributions such as the cases in which a node has multiple possible parents. For added compatibility, one probability distribution can be stored in multiple formats (*i.e.* multiple `DeclarativeDistribution` individuals for the same random variable). Listing A.12 presents a declarative distribution for the random variable `livesAt(person)` represented by the application-dependent format for local probability distribution defined in UnBBayes. The distribution is uniform over the possible values of the random variable (individuals of the class `Address`). See `RandomVariable` in Section A.1 for more information about this random variable.

Listing A.12: Declarative distribution example for random variable `livesAt(person)`

```

1 Individual: RV.livesAt .DD. dist1
2   Types:
3     pr-owl2:DeclarativeDistribution
4   Facts:
5     pr-owl2:isProbabilityDistributionOf  RV.livesAt ,
6     pr-owl2:hasDeclaration  "[_Uniform();_]"^^xsd:string ,
7     pr-owl2:isRepresentedAs  "UnBBayes"^^xsd:string

```

Listing A.13 presents a declarative distribution for the random variable `hasAnnualIncome(person)` represented by the application-dependent format for local probability distribution defined in UnBBayes. The distribution is normal with mean 50,000.00 and standard deviation 20,000.00. See `RandomVariable` in Section A.1 for more information about this random variable.

Listing A.13: Declarative distribution example for random variable `hasAnnualIncome(person)`

```

1 Individual: RV.hasAnnualIncome .DD. dist1
2   Types:
3     pr-owl2:DeclarativeDistribution
4   Facts:
5     pr-owl2:isProbabilityDistributionOf  RV.hasAnnualIncome ,
6     pr-owl2:isRepresentedAs  "UnBBayes"^^xsd:string ,
7     pr-owl2:hasDeclaration  "[_Normal(50000,20000);_]"^^xsd:string

```

`PR-OWLTable` has all the probability assignments for each state of a random variable or resident node stored in a `xsd:decimal` format (future implementations might use a specific data type that represents probabilities). Figure A.27 presents the OWL restrictions for this class.

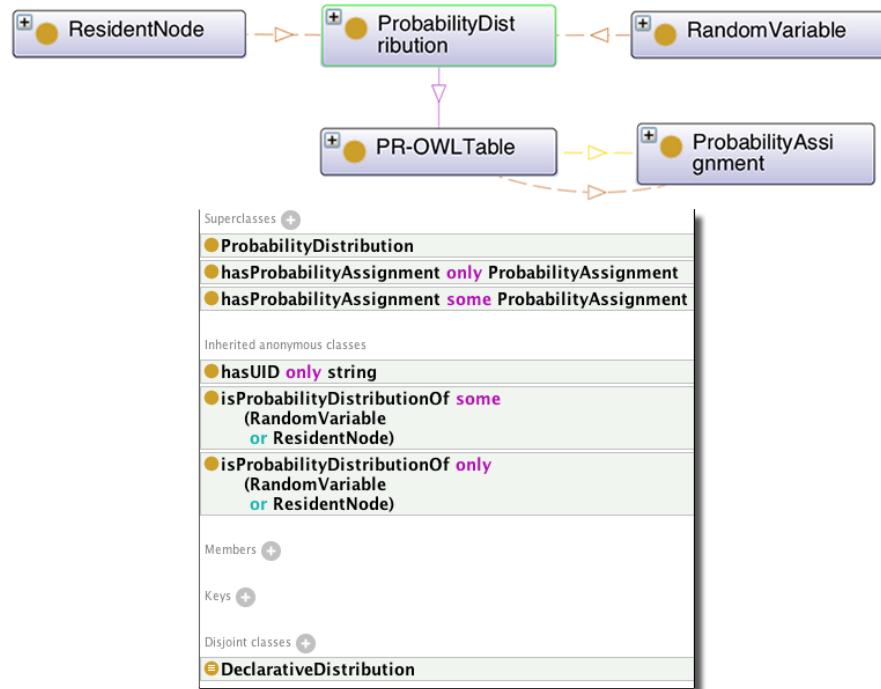


Figure A.27: The OWL restrictions of the PR-OWLTable class.

This format for storing probability distributions cannot represent complex cases for which only formulas can represent a probability distribution (*e.g.* a node that has a variable number of parents) and usually incurs in huge ontologies, since each table can have many cells and each cell is an individual of the **ProbabilityAssignment** class. Therefore, PR-OWL tables are only recommended for the simplest models in which the maximum level of compatibility is desired.

Table A.1: Table representing the distribution for the random variable `isRelated(person1, person2)`.

State	Probability
True	0.001
False	0.999
Absurd	0.0

Listing A.14 presents a PR-OWL table for the random variable `isRelated(person1, person2)`. See `BooleanRandomVariable` in Section A.1 for more information about this random variable. Since the random variable has 3 states and no parents, the number of probability assignments is 3 (see `ProbabilityAssignment` in this Section for more details). Table A.1³ presents the probability distribution this PR-OWL table represents.

Listing A.14: PR-OWL table example for random variable `isRelated(person1, person2)`

```

1 Individual: RV.isRelated.PT.dist1
2   Types:
3     pr-owl2:PR-OWLTable
4   Facts:
5     pr-owl2:isProbabilityDistributionOf RV.isRelated ,
6     pr-owl2:hasProbabilityAssignment RV.isRelated.PT.dist1.PA.assign1 ,
7     pr-owl2:hasProbabilityAssignment RV.isRelated.PT.dist1.PA.assign2 ,
8     pr-owl2:hasProbabilityAssignment RV.isRelated.PT.dist1.PA.assign3

```

³This table follows UnBBayes default way of representing probability distributions, where the states of the child are represented in each row and the combination of parent states are represented on the header of the table. In UnBBayes the rows have to sum to 1.

Table A.2: Table representing the distribution for the domain resident node `isRelated(person1, person2)`.

<code>equalTo(livesAt(person1), livesAt(person2))</code>	True	False	Absurd
True	0.9	0.001	0.0
False	0.1	0.999	0.0
Absurd	0.0	0.0	1.0

Listing A.15: PR-OWL table example for resident node `isRelated(person1, person2)`

```

1 Individual: Mfrag . PersonalInformation . DRN . isRelated . PT . dist1
2   Types:
3     pr-owl2:PR-OWLTable
4   Facts:
5     pr-owl2:hasProbabilityAssignment
6       Mfrag . PersonalInformation . DRN . isRelated . PT . dist1 . PA . assign1 ,
7     pr-owl2:hasProbabilityAssignment
8       Mfrag . PersonalInformation . DRN . isRelated . PT . dist1 . PA . assign2 ,
9     pr-owl2:isProbabilityDistributionOf
10    Mfrag . PersonalInformation . DRN . isRelated ,
11    pr-owl2:hasProbabilityAssignment
12      Mfrag . PersonalInformation . DRN . isRelated . PT . dist1 . PA . assign2 ,
13    pr-owl2:hasProbabilityAssignment
14      Mfrag . PersonalInformation . DRN . isRelated . PT . dist1 . PA . assign4 ,
15    pr-owl2:hasProbabilityAssignment
16      Mfrag . PersonalInformation . DRN . isRelated . PT . dist1 . PA . assign5 ,
17    pr-owl2:hasProbabilityAssignment
18      Mfrag . PersonalInformation . DRN . isRelated . PT . dist1 . PA . assign6 ,
19    pr-owl2:hasProbabilityAssignment
20      Mfrag . PersonalInformation . DRN . isRelated . PT . dist1 . PA . assign7 ,
21    pr-owl2:hasProbabilityAssignment
22      Mfrag . PersonalInformation . DRN . isRelated . PT . dist1 . PA . assign8 ,
23    pr-owl2:hasProbabilityAssignment
24      Mfrag . PersonalInformation . DRN . isRelated . PT . dist1 . PA . assign9

```

Listing A.15 presents a PR-OWL table for the domain resident node `isRelated(person1, person2)`. See `DomainResidentNode` in this Section for more information about this resident node. The distribution is conditional on the parent

input node `equalTo(livesAt(person1), livesAt(person2))`. Since the parent has 3 states and the child has 3 states, the number of probability assignments is 9 (see `ProbabilityAssignment` in this Section for more details). Table A.2⁴ presents the probability distribution this PR-OWL table represents.

ProbabilityAssignment Each cell in an PR-OWL table has a probability assignment for the state of a random variable or resident node given the states of its parent nodes (random variables do not have parent nodes). Thus, the resulting relationship is n-ary and it is represented via the object property `hasProbabilityAssignment` that includes the name of the state to which the probability is being assigned (via the data property `hasStateName`), the probability value itself (via the data property `hasStateProbability`), and the list of states of parent nodes (via the object property `hasConditioningState`) that collectively define the context in which that probability assignment is valid. Also, individuals of the `ProbabilityAssignment` class have the object property `isProbabilityAssignmentIn` that links them with its respective PR-OWL table. Figure A.28 presents the OWL restrictions for this class.

Listing A.16 presents the probability assignments of the PR-OWL table defined for the random variable `isRelated(person1, person2)`. See `PR-OWLTabe` in this Section for more information about this table. Since the random variable has 3 states and no parents, the number of probability assignments is 3.

⁴This table follows UnBBayes default way of representing probability distributions, where the states of the child are represented in each row and the combination of parent states are represented on the header of the table. In UnBBayes the rows have to sum to 1.

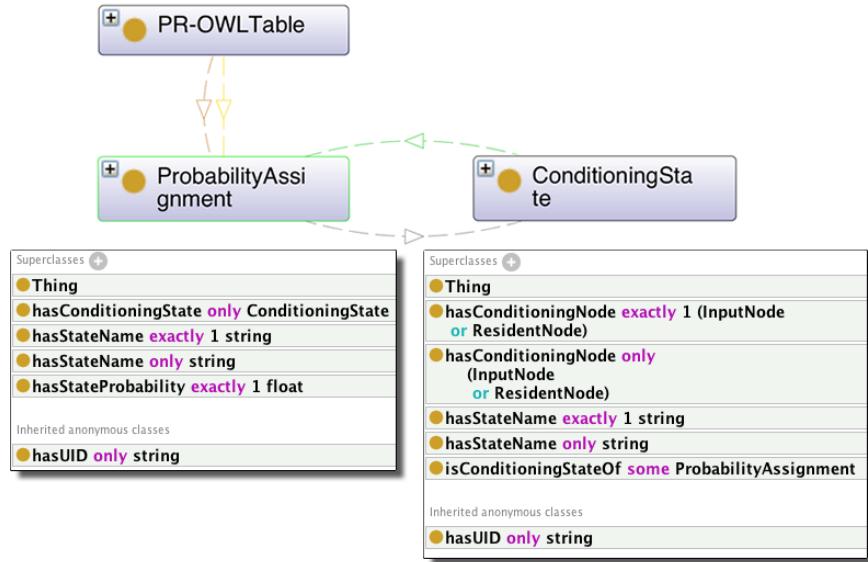


Figure A.28: The OWL restrictions of the classes `ProbabilityAssignment` and `ConditioningState`.

Listing A.16: PR-OWL table probability assignments example for random variable `isRelated(person1, person2)`

```

1 Individual: RV.isRelated.PT.dist1.PA.assign1
2   Types:
3     pr-owl2:ProbabilityAssignment
4   Facts:
5     pr-owl2:isProbabilityAssignmentIn  RV.isRelated.PT.dist1 ,
6     pr-owl2:hasStateName  "true"^^xsd:string ,
7     pr-owl2:hasStateProbability  .001f
8
9 Individual: RV.isRelated.PT.dist1.PA.assign2
10  Types:
11    pr-owl2:ProbabilityAssignment
12  Facts:
13    pr-owl2:isProbabilityAssignmentIn  RV.isRelated.PT.dist1 ,
14    pr-owl2:hasStateName  "false"^^xsd:string ,
15    pr-owl2:hasStateProbability  .999f
16
17 Individual: RV.isRelated.PT.dist1.PA.assign3
18  Types:
19    pr-owl2:ProbabilityAssignment
20  Facts:
21    pr-owl2:isProbabilityAssignmentIn  RV.isRelated.PT.dist1 ,
22    pr-owl2:hasStateName  "absurd"^^xsd:string ,
23    pr-owl2:hasStateProbability  0f

```

Listing A.17: PR-OWL table probability assignments example for domain resident node `isRelated(person1, person2)`

```

1 Individual: MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign1
2   Types:
3     pr-owl2:ProbabilityAssignment
4   Facts:
5     pr-owl2:hasConditioningState
6       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond1 ,
7     pr-owl2:isProbabilityAssignmentIn
8       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 ,
9     pr-owl2:hasStateName "true"^^xsd:string ,
10    pr-owl2:hasStateProbability .9 f
11
12 Individual: MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign2
13   Types:
14     pr-owl2:ProbabilityAssignment
15   Facts:
16     pr-owl2:hasConditioningState
17       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond1 ,
18     pr-owl2:isProbabilityAssignmentIn
19       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 ,
20     pr-owl2:hasStateName "false"^^xsd:string ,
21    pr-owl2:hasStateProbability .1 f
22
23 Individual: MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign3
24   Types:
25     pr-owl2:ProbabilityAssignment
26   Facts:
27     pr-owl2:hasConditioningState
28       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond1 ,
29     pr-owl2:isProbabilityAssignmentIn
30       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 ,
31     pr-owl2:hasStateName "absurd"^^xsd:string ,
32    pr-owl2:hasStateProbability 0f
33
34 Individual: MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign4
35   Types:
36     pr-owl2:ProbabilityAssignment
37   Facts:
38     pr-owl2:hasConditioningState
39       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond2 ,
40     pr-owl2:isProbabilityAssignmentIn
41       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 ,
42     pr-owl2:hasStateName "true"^^xsd:string ,
43    pr-owl2:hasStateProbability .001 f
44
45 Individual: MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign5
46   Types:
47     pr-owl2:ProbabilityAssignment
48   Facts:
49     pr-owl2:hasConditioningState
50       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond2 ,
51     pr-owl2:isProbabilityAssignmentIn
52       MFrag. PersonalInformation .DRN. isRelated .PT. dist1 ,
53     pr-owl2:hasStateName "false"^^xsd:string ,

```

```

54   pr-owl2:hasStateProbability .999 f
55
56 Individual: Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign6
57 Types:
58   pr-owl2:ProbabilityAssignment
59 Facts:
60   pr-owl2:hasConditioningState
61     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond2 ,
62   pr-owl2:isProbabilityAssignmentIn
63     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 ,
64   pr-owl2:hasStateName "absurd"^^xsd:string ,
65   pr-owl2:hasStateProbability 0f
66
67 Individual: Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign7
68 Types:
69   pr-owl2:ProbabilityAssignment
70 Facts:
71   pr-owl2:hasConditioningState
72     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond3 ,
73   pr-owl2:isProbabilityAssignmentIn
74     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 ,
75
76   pr-owl2:hasStateName "true"^^xsd:string ,
77   pr-owl2:hasStateProbability 0f
78
79 Individual: Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign8
80 Types:
81   pr-owl2:ProbabilityAssignment
82 Facts:
83   pr-owl2:hasConditioningState
84     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond3 ,
85   pr-owl2:isProbabilityAssignmentIn
86     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 ,
87   pr-owl2:hasStateName "false"^^xsd:string ,
88   pr-owl2:hasStateProbability 0f
89
90 Individual: Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign9
91 Types:
92   pr-owl2:ProbabilityAssignment
93 Facts:
94   pr-owl2:hasConditioningState
95     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond3 ,
96   pr-owl2:isProbabilityAssignmentIn
97     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 ,
98   pr-owl2:hasStateName "absurd"^^xsd:string ,
99   pr-owl2:hasStateProbability 1f

```

Listing A.17 presents the probability assignments of the PR-OWL table defined for the domain resident node `isRelated(person1, person2)`. See `PR-OWLTABLE` in this Section for more information about this table. Since the parent has 3 states and the

child has 3 states, the number of probability assignments is 9. For more information on the conditioning states for these probability assignments, see `ConditioningState` in this Section.

`ConditioningState` represents the conditioning state for a probability assignment. In other words, it defines the state of a parent that is conditioning the probability assignment for a specific state of the child node. Individuals of this class are used to build PR-OWL probabilistic distribution tables. Each cell of such a table corresponds to a probability assignment of a possible value of a node given one combination of the states of its parents. Each individual of class `ConditioningState` represents one parent/state pair, so a probability assignment is conditioned by a set of `ConditioningState` pairs (one for each parent node). Figure A.28 presents the OWL restrictions for this class.

Listing A.18 presents the conditioning states of the probability assignments of the PR-OWL table defined for the domain resident node `isRelated(person1, person2)`. See `PR-OWLTable` and `ProbabilityAssignment` in this Section for more information about this table. Since the parent has 3 states the number of conditioning states is also 3 (conditioning the parent on the state true, false, and absurd).

Listing A.18: Conditioning states for the probability assignments of the PR-OWL table defined for the domain resident node `isRelated(person1, person2)`

```

1 Individual: Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond1
2 Types:
3   pr-owl2:ConditioningState
4 Facts:
5   pr-owl2:isConditioningStateOf
6     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign1 ,
7   pr-owl2:isConditioningStateOf
8     Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign2 ,
9   pr-owl2:isConditioningStateOf
10    Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign3 ,
11   pr-owl2:hasConditioningNode  Mfrag. PersonalInformation .GIN. equalTo1 ,
12   pr-owl2:hasStateName  "true"^^xsd:string
13
14 Individual: Mfrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond2
15 Types:
16   pr-owl2:ConditioningState
17 Facts:
```

```

18 pr-owl2:isConditioningStateOf MFrag. PersonalInformation .DRN. isRelated .PT.
19     dist1 .PA. assign4 ,
20 pr-owl2:isConditioningStateOf MFrag. PersonalInformation .DRN. isRelated .PT.
21     dist1 .PA. assign5 ,
22 pr-owl2:isConditioningStateOf MFrag. PersonalInformation .DRN. isRelated .PT.
23     dist1 .PA. assign6 ,
24 pr-owl2:hasConditioningNode MFrag. PersonalInformation .GIN. equalTo1 ,
25 pr-owl2:hasStateName " false " ^ xsd:string
26
27 Individual: MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .CS. cond3
28 Types:
29     pr-owl2:ConditioningState
30 Facts:
31     pr-owl2:isConditioningStateOf
32         MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign7 ,
33     pr-owl2:isConditioningStateOf
34         MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign8 ,
35     pr-owl2:isConditioningStateOf
36         MFrag. PersonalInformation .DRN. isRelated .PT. dist1 .PA. assign9 ,
37     pr-owl2:hasConditioningNode MFrag. PersonalInformation .GIN. equalTo1 ,
38     pr-owl2:hasStateName " absurd " ^ xsd:string

```

A.3 MEBN Expressions

Figure A.29 presents the general hierarchy of the classes used for defining arguments and MEBN expressions. This is just part of the hierarchy shown in Figure A.1.

Figure A.30 presents the main concepts and their relations necessary for defining arguments and MEBN expressions. There are mainly two different types of arguments:

1. those used for mapping random variable arguments to OWL properties domain or range (represented by the class `MappingArgument`), and
2. those used in MEBN expressions (`MExpressionArguemnt`, `ExemplarArgument`, `OrdinalVariableArgument`, and `ConstantArgument`).

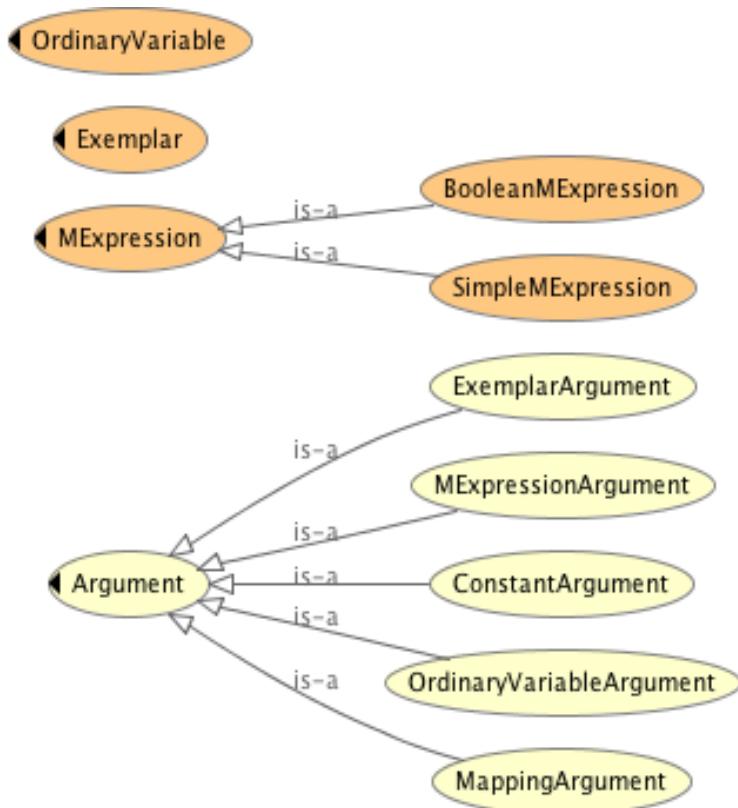


Figure A.29: The hierarchy of the main classes for representing arguments and MEBN expressions.

A `MExpression` has a type of random variable as its main element (*e.g.*, `equalTo`, `or`, and, `livesAt`, etc). Every node is represented by an MEBN expression. However, some nodes can only be represented by a specific type of `MExpression`. `FindingResidentNode`, `FindingInputNode`, and `ContextNode` can only be represented by `BooleanMExpression`. A `BooleanMExpression` only allow a specific type of random variable, `BooleanRandomVariable`. `DomainResidentNode` can only be represented by `SimpleMExpression`, which is an MEBN expression that only has arguments of type `OrdinaryVariable` or `ConstantArgument`.

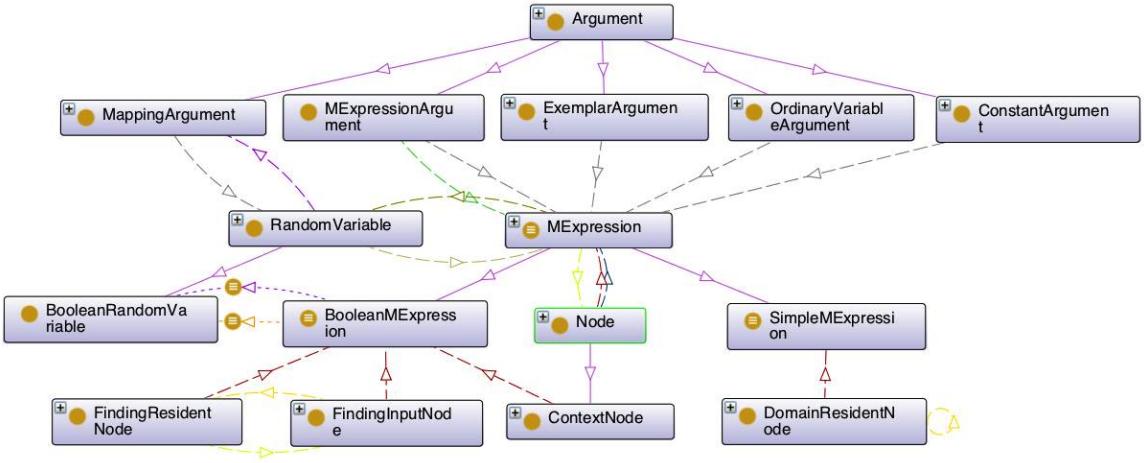


Figure A.30: Graph with main concepts and their relations necessary for defining arguments and MEBN expressions.

MEExpression represents a first-order logic formula or term, which has the random variable (RV) as its main element. The number of arguments defined on the MEBN expression has to be the same as the number of arguments defined on the RV it refers to. Furthermore, the argument types have to be compatible. Figure A.31 presents the OWL restrictions for this class.

There are basically four types of arguments which can be used to express a MEBN expression: **ConstantArgument**, **OrdinaryVariableArgument**, **ExemplarArgument**, and **MExpressionArgument**. They will be explained latter in this Section.

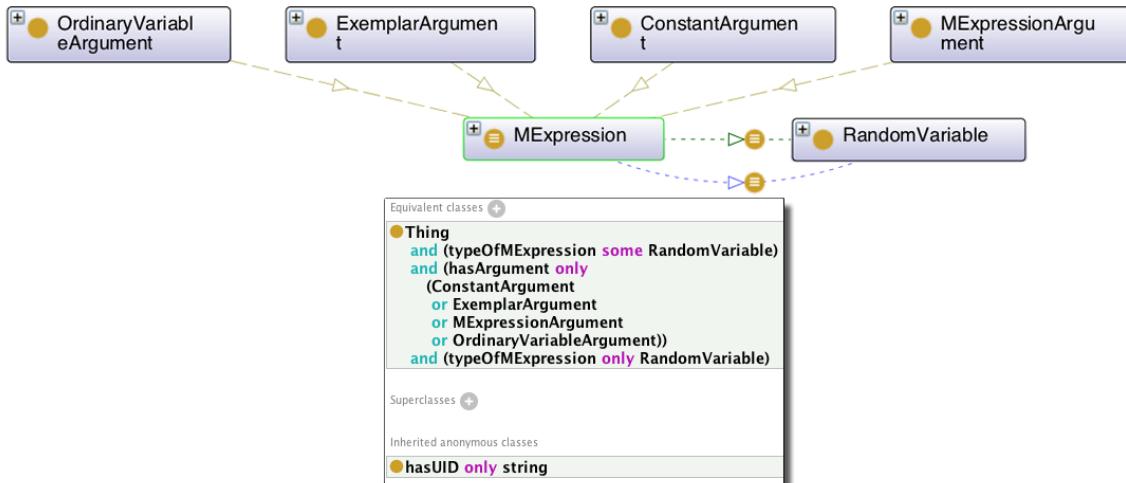


Figure A.31: The OWL restrictions of the **MExpression** class.

SimpleMExpression represents an atomic formula or term, also called atom. Therefore, it can only have arguments of type **ConstantArgument** or **OrdinaryVariableArgument**, in other words, it cannot have subformulas/subterms. Figure A.32 presents the OWL restrictions for this class.

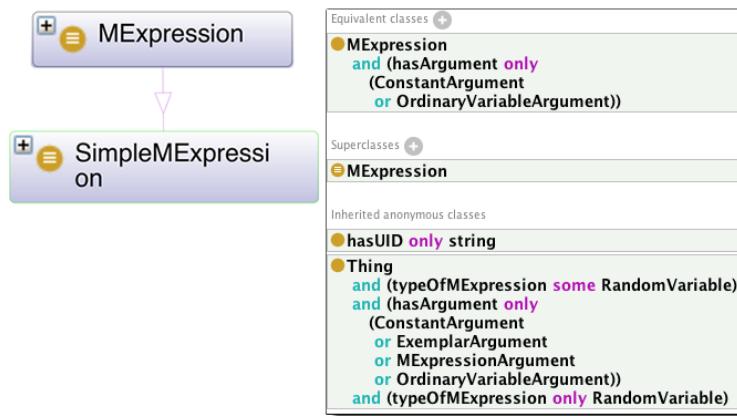


Figure A.32: The OWL restrictions of the **SimpleMExpression** class.

Listing A.19 presents a simple MEBN expression for the resident node `isRelated(person1, person2)` (see `DomainResidentNode` in Section A.2 for more information on this node). The type of formula for this MEBN expression is the random variable `isRelated` (see `BooleanRandomVariable` in Section A.1 for more information on this random variable). Finally, it has two ordinary variable arguments, `person1` and `person2` (see `OrdinaryVariableArgument` in this Section for more information about these arguments).

Listing A.19: Simple MEBN expression example

```

1 Individual: Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated
2   Types:
3     pr-owl2:SimpleMExpression
4   Facts:
5     pr-owl2:typeOfMExpression  RV. isRelated ,
6     pr-owl2:isMExpressionOf  Mfrag. PersonalInformation .DRN. isRelated ,
7     pr-owl2:hasArgument
8       Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated .OVA. person1 ,
9     pr-owl2:hasArgument
10      Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated .OVA. person2

```

`BooleanMExpression` represents a Boolean formula. In other words, the random variable (RV) which defines the type of MEBN expression can only be a `BooleanRandomVariable`. Figure A.33 presents the OWL restrictions for this class.

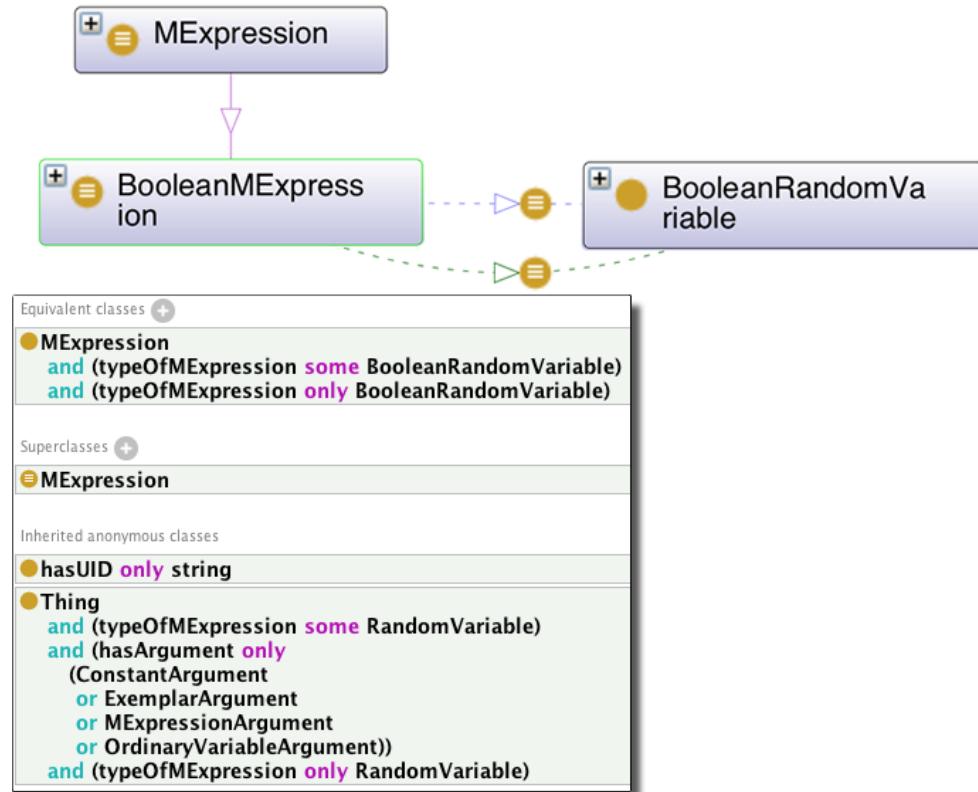


Figure A.33: The OWL restrictions of the **BooleanMExpression** class.

Listing A.20 presents the Boolean MEBN expression `equalTo(hasAnnualIncome(Bill), 75,000.00)`. It has two arguments, the MEBN expression `hasAnnualIncome(Bill)` (see `MExpressionArgument` in this Section for more information) and the data constant `75,000.00` (see `ConstantArgument` in this Section for more information). The type of random variable for this expression is `equalTo` (see `BooleanRandomVariable` in Section A.1 for more information). This MEBN expression defines the finding resident node that states that Bill has an annual income of `75,000.00` (see `FindingResidentNode` in Section A.2 for more information about this finding resident node).

Listing A.20: Example of Boolean MEBN expression

```

1 Individual: MFragment.Finding1.FRN.equalTo1.BME.equalTo1
2   Types:
3     pr-owl2:BooleanMExpression
4   Facts:
5     pr-owl2:typeOfMExpression pr-owl2:equalTo ,
6     pr-owl2:isMExpressionOf MFragment.Finding1.FRN.equalTo1 ,
7     pr-owl2:hasArgument MFragment.Finding1.FRN.equalTo1.BME.equalTo1.MA.arg1
8     pr-owl2:hasArgument MFragment.Finding1.FRN.equalTo1.BME.equalTo1.CA.float1 ,

```

`Argument` is an argument used for either constructing MEBN expressions (through the use of `ConstantArgument`, `OrdinaryVariableArgument`, `ExemplarArgument`, and `MExpressionArgument`) or for mapping random variables arguments defined in PR-OWL to domain and ranges of properties defined in OWL (through the use of `MappingArgument`). Figure A.34 presents the OWL restrictions for this class.

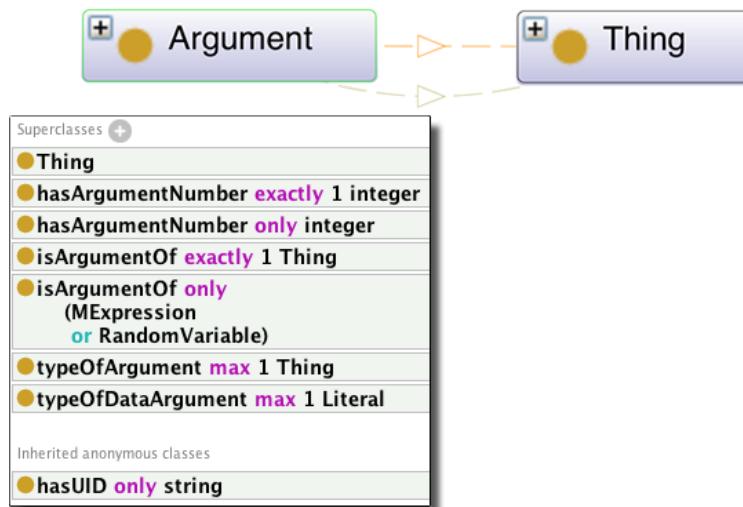


Figure A.34: The OWL restrictions of the `Argument` class.

`ConstantArgument` is used to represent formulas or terms which use either data and/or object constants, *e.g.*, `equalTo(livesAt(Bill), address1)` (where

Bill and address1 are constants, which represent Person and Address, respectively), `equalTo(hasAnnualIncome(Bill), 75,000.00)` (assuming income is just a number, which represents value in US Dollar and Bill and 75,000.00 are constants, which represent Person and float, respectively). Figure A.35 presents the OWL restrictions for this class.

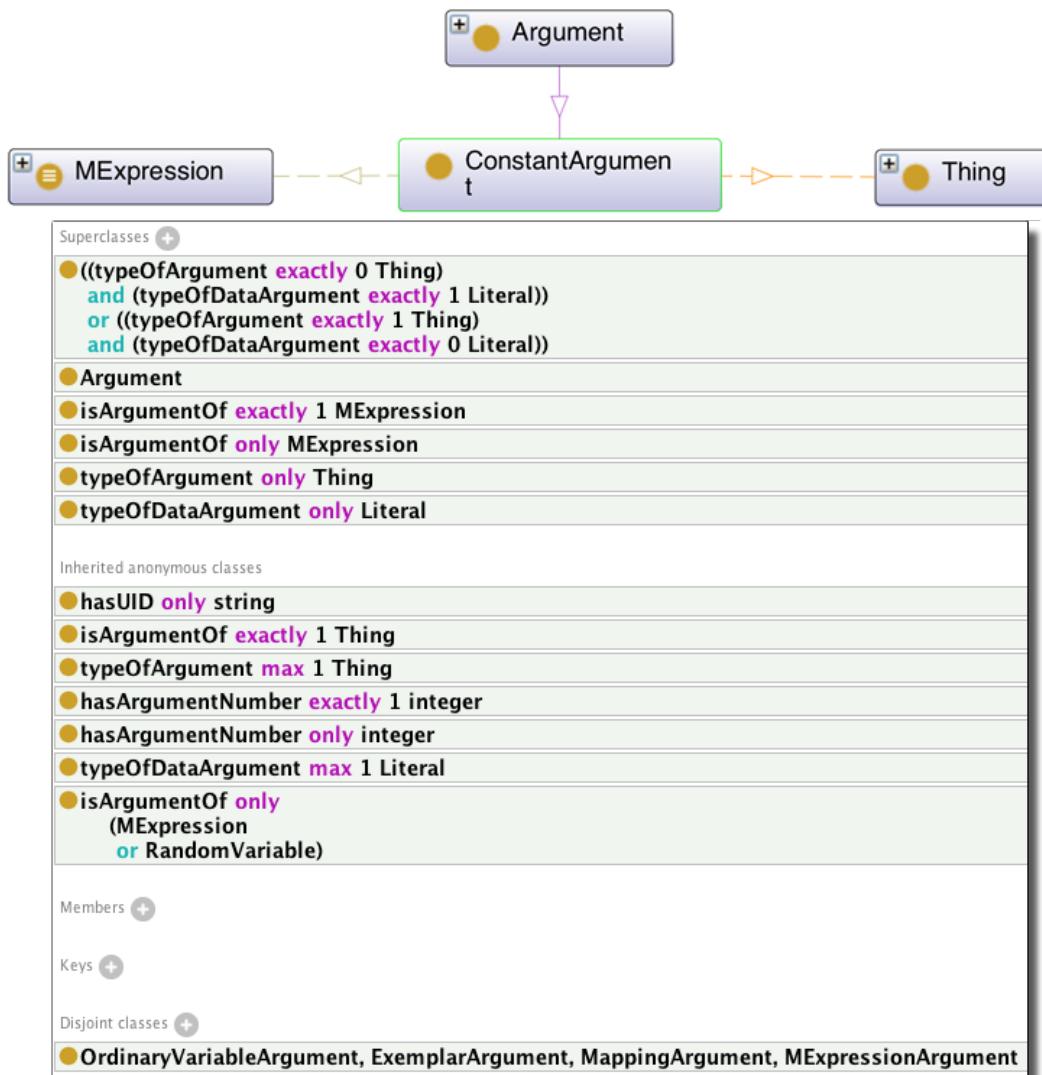


Figure A.35: The OWL restrictions of the `ConstantArgument` class.

Listing A.21 presents two constant arguments. On the one hand, `Bill` is an object argument used on the `MExpression hasAnnualIncome(Bill)`, which is the first argument (see `MExpressionArgument` in this Section for more information). On the other hand, `75,000.00` is a data argument used on the `MExpression equalTo(hasAnnualIncome(Bill), 75,000.00)`, which is the second argument (see `BooleanMExpression` in this Section for more information).

Listing A.21: Example of object and data constant arguments

```

1  <!-- The object constant argument for Bill -->
2 Individual: Mfrag.Finding1.FRN.equalTo1.BME.equalTo1.ME.hasAnnualIncome1.CA.
   person1
3 Types:
4   pr-owl2:ConstantArgument
5 Facts:
6   pr-owl2:isArgumentOf
7     Mfrag.Finding1.FRN.equalTo1.BME.equalTo1.ME.hasAnnualIncome1,
8   pr-owl2:typeOfArgument Bill,
9   pr-owl2:hasArgumentNumber 1
10
11 <!-- The data constant argument for 75,000.00 -->
12 Individual: Mfrag.Finding1.FRN.equalTo1.BME.equalTo1.CA.float1
13 Types:
14   pr-owl2:ConstantArgument
15 Facts:
16   pr-owl2:isArgumentOf Mfrag.Finding1.FRN.equalTo1.BME.equalTo1,
17   pr-owl2:typeOfDataArgument 75000f,
18   pr-owl2:hasArgumentNumber 2

```

`OrdinaryVariableArgument` is used to represent free variable arguments (not quantified over) used in a formula or term, *e.g.*, `livesAt(person)`, where `person` is a free variable, that can be substituted by an individual of the class `Person`.

Figure A.36 presents the OWL restrictions for this class.

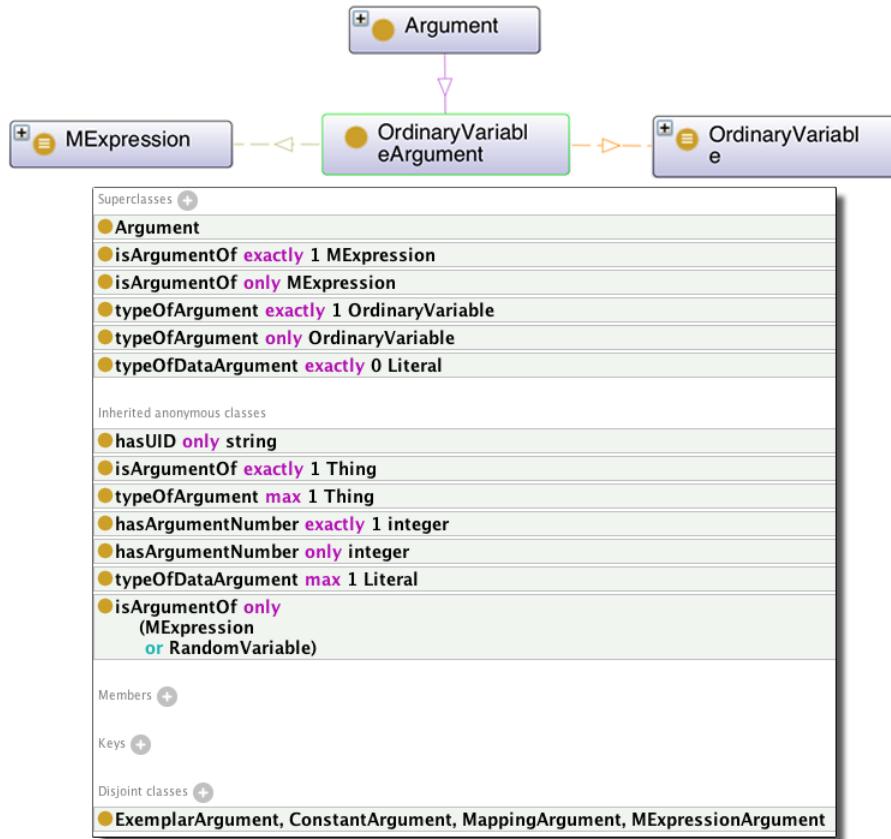


Figure A.36: The OWL restrictions of the `OrdinaryVariableArgument` class.

Listing A.22: Ordinary variable arguments for the simple MEBN expression `isRelated(person1, person2)`

```

1 Individual: Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated .OVA. person1
2   Types:
3     pr-owl2:OrdinaryVariableArgument
4   Facts:
5     pr-owl2:isArgumentOf Mfrag. PersonalInformation .DRN. isRelated .SME.
6       isRelated ,
7     pr-owl2:typeOfArgument Mfrag. PersonalInformation .OV. person1 ,
8     pr-owl2:hasArgumentNumber 1
9 Individual: Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated .OVA. person2
10   Types:
11     pr-owl2:OrdinaryVariableArgument
12   Facts:
13     pr-owl2:isArgumentOf Mfrag. PersonalInformation .DRN. isRelated .SME.
14       isRelated ,
15     pr-owl2:typeOfArgument Mfrag. PersonalInformation .OV. person2 ,
16     pr-owl2:hasArgumentNumber 2

```

Listing A.22 presents the ordinary variable arguments of the simple MEBN expression `isRelated(person1, person2)`. See `SimpleMExpression` in this Section for more information about this expression. The first argument is the ordinary variable `MFrag.PersonalInformation.OV.person1` and the second argument is the ordinary variable `MFrag.PersonalInformation.OV.person2`. See `OrdinaryVariable` in this section for more information about ordinary variables.

`ExemplarArgument` is used to represent a filler for a bound variable (variables that are quantified over) used in a formula, e.g., `forAll(mother)(forAll(child)(implies(hasChild(mother, child), isRelated(child, mother))))`, where `mother` and `child` are bound variables of type `Person`. Figure A.37 presents the OWL restrictions for this class.

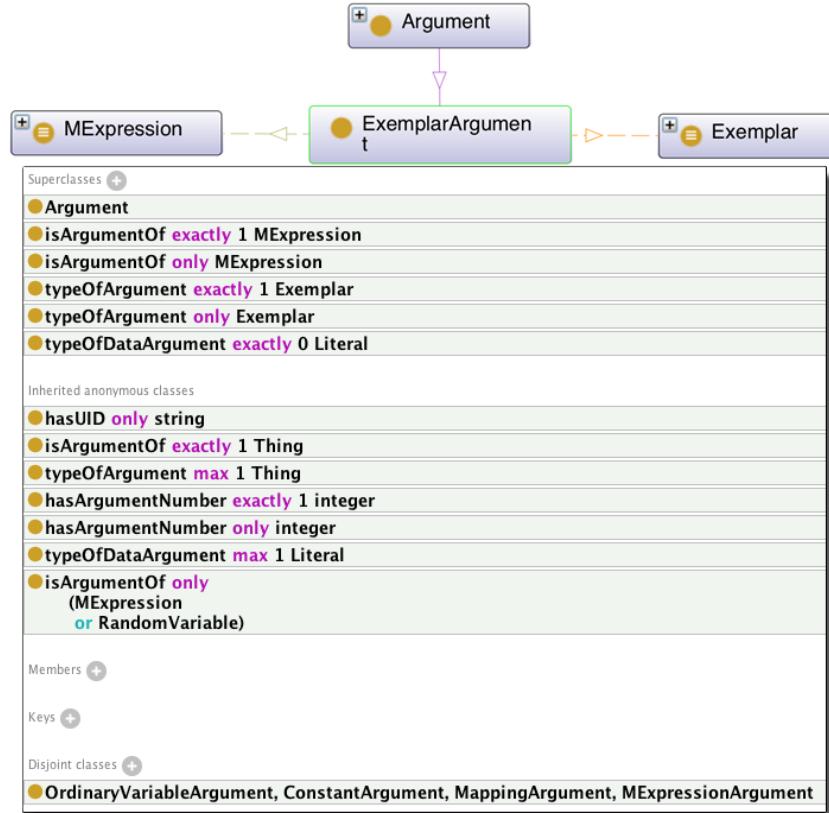


Figure A.37: The OWL restrictions of the `ExemplarArgument` class.

Listing A.23 presents the `mother` and `child` exemplar arguments that are used on both `forAll` statements from the previous example.

Listing A.23: Exemplar argument example

```

1 <!-- The exemplar argument mother -->
2 Individual: Mfrag. PersonalInformation .GIN. forAll1 .BME. forAll1 .EA. mother
3   Types:
4     pr-owl2:ExemplarArgument
5   Facts:
6     pr-owl2:isArgumentOf Mfrag. PersonalInformation .GIN. forAll1 .BME. forAll1 ,
7     pr-owl2:typeOfArgument Mfrag. PersonalInformation .E. mother ,
8     pr-owl2:hasArgumentNumber "1"^^xsd:int
9
10 <!-- The exemplar argument child -->
11 Individual: Mfrag. PersonalInformation .GIN. forAll1 .BME. forAll1 .BME. forAll1 .EA.
12   child
13   Types:
14     pr-owl2:ExemplarArgument
15   Facts:
16     pr-owl2:isArgumentOf Mfrag. PersonalInformation .GIN. forAll1 .BME. forAll1 .
17       BME. forAll1 ,
18     pr-owl2:typeOfArgument Mfrag. PersonalInformation .E. child ,
19     pr-owl2:hasArgumentNumber "1"^^xsd:int

```

`MExpressionArgument` is used to allow the construction of complex formulas or terms (more than one RV used in the formula or term), e.g., `equalTo(livesAt(person1), livesAt(person2))`, where `person1` and `person2` are free variables of type `Person`. Figure A.38 presents the OWL restrictions for this class.

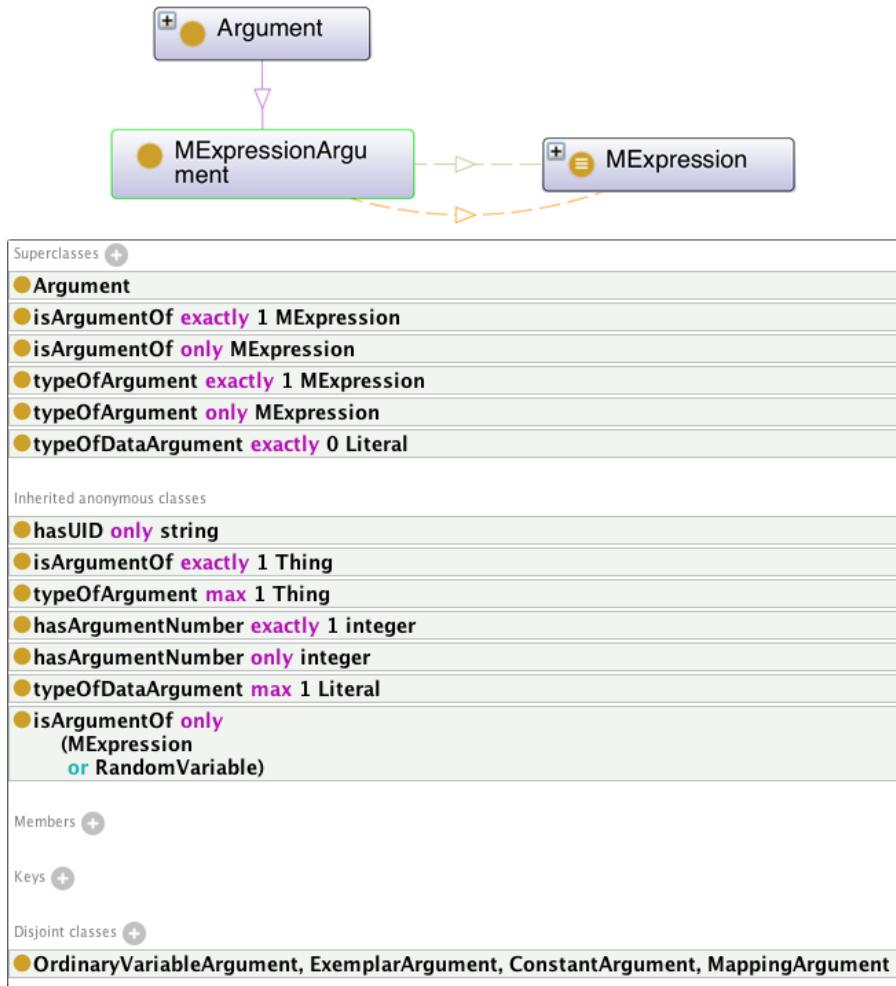


Figure A.38: The OWL restrictions of the **MExpressionArgument** class.

Listing A.24 presents the MEBN expression argument `hasAnnualIncome(Bill)` used on the Boolean MEBN expression `equalTo(hasAnnualIncome(Bill), 75,000.00)` (see `BooleanMEBNExpression` in this Section for more information). The type of this argument is the MEBN expression itself, which has the random variable `hasAnnualIncome` as its type (see `RandomVariable` in Section A.1 for more information). The only argument for this expression is the object constant `Bill` (see `ConstantArgument` in this Section for more information).

Listing A.24: Example of MEBN expression argument

```

1 <!-- The MEBN expression argument -->
2 Individual: Mfrag.Finding1.FRN.equalTo1.BME.equalTo1.MA.arg1
3 Types:
4   pr-owl2:MExpressionArgument
5 Facts:
6   pr-owl2:isArgumentOf Mfrag.Finding1.FRN.equalTo1.BME.equalTo1,
7   pr-owl2:typeOfArgument
8     Mfrag.Finding1.FRN.equalTo1.BME.equalTo1.ME.hasAnnualIncome1,
9   pr-owl2:hasArgumentNumber 1
10
11 </owl:NamedIndividual>
12
13 <!-- The MEBN expression used as an argument -->
14 Individual: Mfrag.Finding1.FRN.equalTo1.BME.equalTo1.ME.hasAnnualIncome1
15 Types:
16   pr-owl2:MExpression
17 Facts:
18   pr-owl2:isTypeOfArgumentIn
19     Mfrag.Finding1.FRN.equalTo1.BME.equalTo1.MA.arg1,
20   pr-owl2:hasArgument
21     Mfrag.Finding1.FRN.equalTo1.BME.equalTo1.ME.hasAnnualIncome1.CA.person1

```

`MappingArgument` is used to map random variable arguments to OWL properties domain or range. On the one hand, to map a random variable argument to a domain of some OWL property, it is necessary to say that this argument `isSubjectIn` some OWL property. On the other hand, to map a random variable argument to a range of some OWL property, it is necessary to say that this argument `isObjectIn` some OWL property. In both cases, the type of the argument has to be consistent with the OWL property it points to. If `isSubjectIn` is used, then the type of the argument (defined by the property `isSubstitutedBy`) has to be the same as the domain of the property it points to. However, if `isObjectIn` is used instead, then the type of the argument (defined by the property `isSubstitutedBy`) has to be the same as the range of the property it points to. Since OWL DL does not allow the use of restricted vocabulary when defining the ontology, instead of saying that `isSubjectIn` only `rdf:Property` and `isObjectIn` only `rdf:Property`, the restrictions are defined as `isSubjectIn` only `rdfs:anyURI` and `isObjectIn` only `rdfs:anyURI` (see Section 4.4.2 for more

information). Nevertheless, the semantics of PR-OWL enforce that these URIs have to point to RDF properties. The same reasoning applies to the restriction `isSubstitutedBy only/some rdfs:anyURI`. The semantics of PR-OWL enforce that these URIs have to point to either classes or data types. Figure A.39 presents the OWL restrictions for this class.

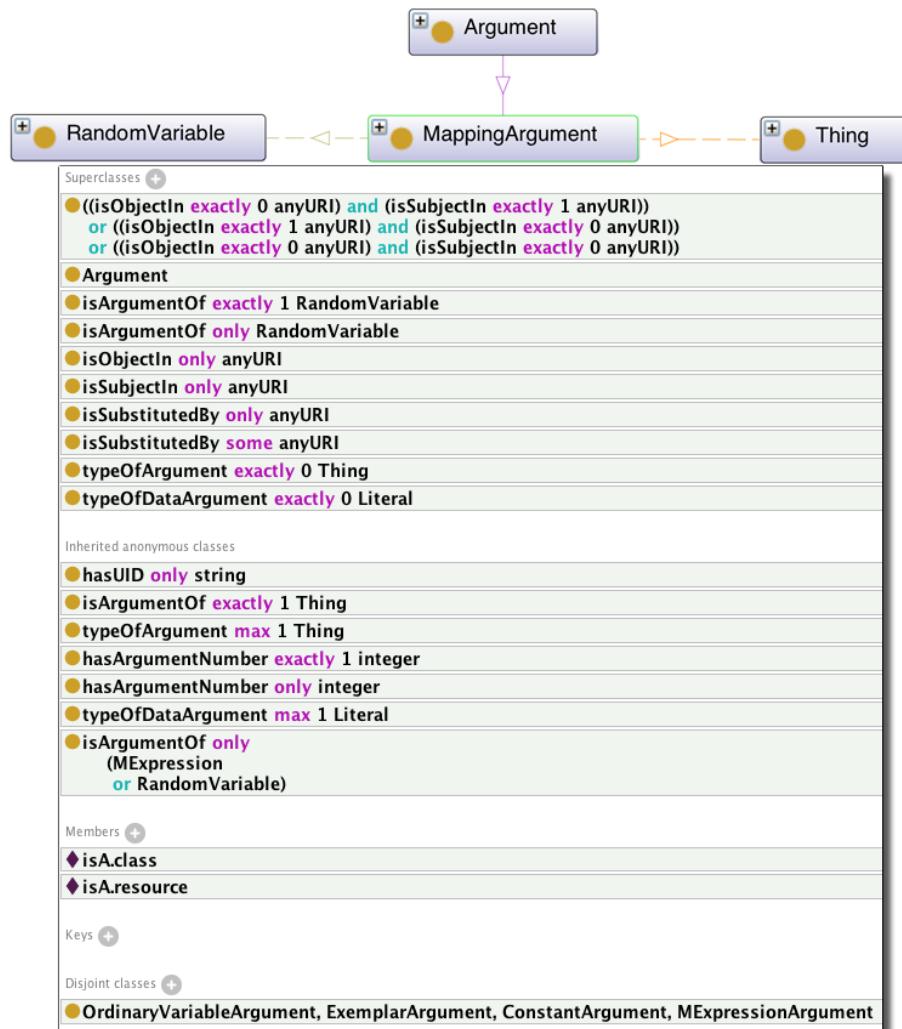


Figure A.39: The OWL restrictions of the `MappingArgument` class.

Listing A.25 presents the mapping argument for the random variable `hasAnnualIncome(person)`. It maps the first and only argument `person` to the subject of the OWL property `hasAnnualIncome`. See Section A.1 for more information on this random variable.

Listing A.25: Example of mapping argument

```

1 Individual: RV.hasAnnualIncome.MA.person
2 Types:
3   pr-owl2:MappingArgument
4 Facts:
5   pr-owl2:isArgumentOf  RV.hasAnnualIncome ,
6   pr-owl2:isSubjectIn    "&ex; hasAnnualIncome"^^xsd:anyURI ,
7   pr-owl2:hasArgumentNumber  1

```

`OrdinaryVariable` is a placeholder used in MFrags to refer to non-specific entities as arguments in MEBN expressions in its MFragment. It is used to represent free variables (not quantified over) used in a formula or term, *e.g.*, `livesAt(person)`, where `person` is a free variable, that can be substituted by an individual of the class `Person`. Figure A.40 presents the OWL restrictions for this class.

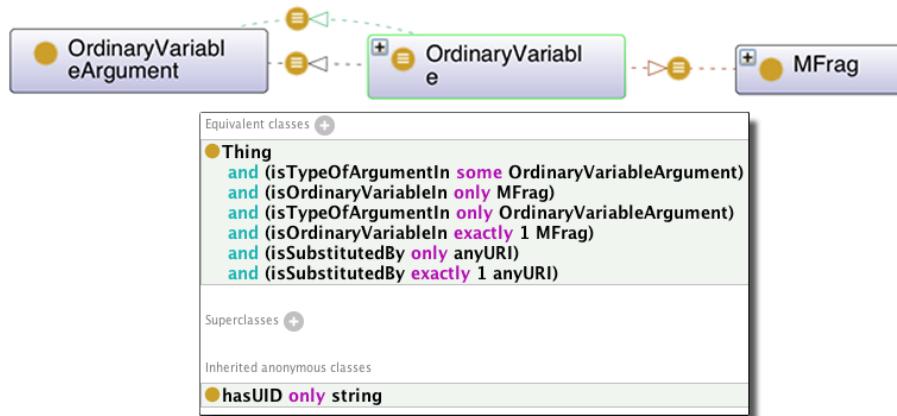


Figure A.40: The OWL restrictions of the `OrdinaryVariable` class.

Listing A.26 presents an ordinary variable `person1`, which can be substituted by individuals of the class `Person`. This ordinary variable is defined in the MFragment `PersonalInformation`. Finally, this ordinary variable is used as an argument in three different MEBN expressions (see `OrdinaryVariableArgument` in this Section for more information).

Listing A.26: Example of ordinary variable

```

1 Individual: Mfrag. PersonalInformation .OV. person1
2   Types:
3     pr-owl2:OrdinaryVariable
4   Facts:
5     pr-owl2:isOrdinaryVariableIn  Mfrag. PersonalInformation ,
6     pr-owl2:isTypeOfArgumentIn
7       Mfrag. PersonalInformation .CN. not1 .BME. not1 .BME. equalTo1 .OVA. person1 ,
8     pr-owl2:isTypeOfArgumentIn
9       Mfrag. PersonalInformation .GIN. equalTo1 .BME. equalTo1 .ME. livesAt1 .OVA.
10    person1 ,
11    pr-owl2:isTypeOfArgumentIn
12      Mfrag. PersonalInformation .DRN. isRelated .SME. isRelated .OVA. person1 ,
pr-owl2:isSubstitutedBy "&ex;Person"^^xsd:anyURI

```

`Exemplar` represents an exemplar constant in an MEBN quantifier expression. Each MEBN quantifier expression corresponds to a first-order formula beginning with a universal or existential quantifier. The exemplar constant in the MEBN expression represents a generic individual within the scope of the universal or existential quantifier of the corresponding first-order formula. It is used to represent bound variables (variables that are quantified over) used in a formula, *e.g.*, `forAll(mother)(forAll(child)(implies(hasChild(mother,child), isRelated(child,mother))))`, where `mother` and `child` are bound variables of type `Person`. Figure A.41 presents the OWL restrictions for this class.

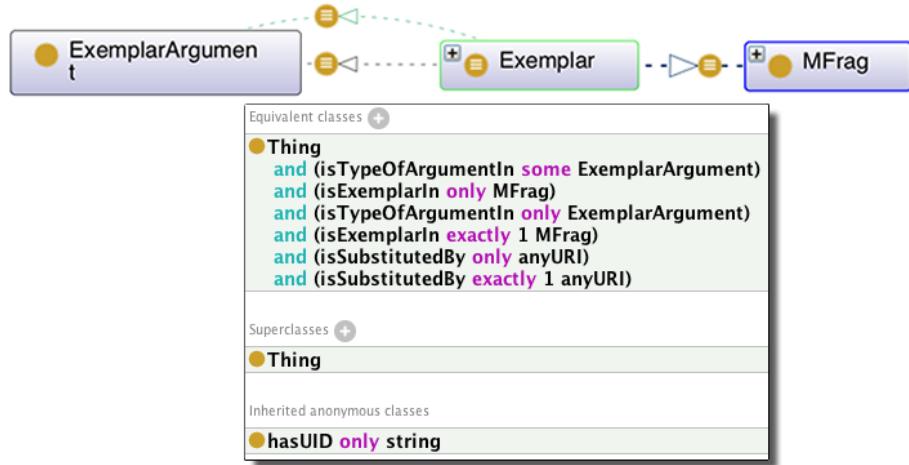


Figure A.41: The OWL restrictions of the `Exemplar` class.

Listing A.27 presents the exemplar variables `mother` and `child` used on the `forAll` formulas on the previous example.

Listing A.27: Exemplar example

```

1  <!-- The exemplar variable child -->
2  Individual: Mfrag. PersonalInformation .E. child
3  Types:
4      pr-owl2:Exemplar
5  Facts:
6      pr-owl2:isExemplarIn Mfrag. PersonalInformation ,
7          pr-owl2:isTypeOfArgumentIn Mfrag. PersonalInformation .GIN. forAll1 .BME.
8              forAll1 .BME. forAll1 .BME. implies1 .ME. hasChild1 .EA. child ,
9              pr-owl2:isTypeOfArgumentIn Mfrag. PersonalInformation .GIN. forAll1 .BME.
10                 forAll1 .BME. forAll1 .EA. child ,
11                 pr-owl2:isTypeOfArgumentIn Mfrag. PersonalInformation .GIN. forAll1 .BME.
12                     forAll1 .BME. forAll1 .BME. implies1 .ME. isRelated1 .EA. child
13
14 <!-- The exemplar variable mother -->
15 Individual: Mfrag. PersonalInformation .E. mother
16 Types:
17     pr-owl2:Exemplar
18 Facts:
19     pr-owl2:isTypeOfArgumentIn Mfrag. PersonalInformation .GIN. forAll1 .BME.
         forAll1 .EA. mother ,
20     pr-owl2:isExemplarIn Mfrag. PersonalInformation ,
21     pr-owl2:isTypeOfArgumentIn Mfrag. PersonalInformation .GIN. forAll1 .BME.
         forAll1 .BME. forAll1 .BME. implies1 .ME. hasChild1 .EA. mother ,
22     pr-owl2:isTypeOfArgumentIn Mfrag. PersonalInformation .GIN. forAll1 .BME.
         forAll1 .BME. forAll1 .BME. implies1 .ME. isRelated1 .EA. mother

```

Appendix B: Use Cases Implementation Details

In this Appendix I will give the details of the probabilistic ontologies presented in Chapter 5.

B.1 Probabilistic Ontology for Procurement Fraud Detection and Prevention in Brazil

All the assumptions for the RVs created and for defining their LPD will be described for every Mfrag designed for the MTheory that represents the PO for the Procurement Fraud Detection and Prevention implemented. In each Mfrag, the resident RVs are shown as yellow rounded rectangles; the input RVs are shown as gray trapezoids; the context RVs are shown as green pentagons.

In order to make reference easier, the rules defined during Analysis & Design in Chapter 5 Subsection 5.1.2 will be repeated here. The rules are:

1. If a member of the committee has a relative (mother, father, brother, or sister) responsible for a bidder in the procurement, then it is more likely that a relation exists between the committee and the enterprises, which inhibits competition.
2. If a member of the committee lives at the same address as a person responsible for a bidder in the procurement, then it is more likely that a relation exists between the committee and the enterprises, which lowers competition.
3. If a contract of high value related to a procurement has a responsible person of the winner enterprise with low education or low annual income, then this person is likely to be a front for the firm, which lowers competition.
4. If the responsible person of the winner enterprise is also responsible for another enterprise that has its CGC suspended for procuring with the public administration, then this procurement is more likely to need further investigation.

5. If the responsible people for the bidders in the procurement are related to each other, then a competition is more likely to have been compromised.
6. If 1, 2, 3, or 5, then the procurement is more likely to require further investigation.
7. If a member of the committee has been convicted of a crime or has been penalized administratively, then he/she does not have a clean history. If he/she was recently investigated, then it is likely that he/she does not have a clean history.
8. If the relation defined in 1 and 2 is found in previous procurements, then it is more likely that there will be a relation between this committee and future bidders.
9. If 7 or 8, then it is more likely that the committee needs to be changed.

In order to facilitate the understanding of the MFrags in this model, it is useful to know the dependence between the MFrags.

The MFrags with no dependency are:

1. Personal Information;
2. Procurement Information;
3. Enterprise Information; and
4. Judgement History.

The other MFrags have the following dependence:

1. Front of Enterprise
 - (a) Procurement Information; and
 - (b) Personal Information.
2. Exists Front in Enterprise
 - (a) Enterprise Information; and

(b) Front of Enterprise.

3. Related Participant Enterprises

- (a) Procurement Information;
- (b) Enterprise Information; and
- (c) Personal Information.

4. Member Related to Participant

- (a) Personal Information;
- (b) Procurement Information; and
- (c) Enterprise Information.

5. Competition Compromised

- (a) Procurement Information;
- (b) Exists Front in Enterprise;
- (c) Related Participant Enterprises; and
- (d) Member Related to Participant.

6. Related to Previous Participants

- (a) Personal Information;
- (b) Procurement Information; and
- (c) Enterprise Information.

7. Suspicious Committee

- (a) Procurement Information;
- (b) Judgment History; and
- (c) Related to Previous Participants.

8. Owns Suspended Enterprise

(a) Enterprise Information.

9. Suspicious Procurement

(a) Procurement Information;

(b) Enterprise Information;

(c) Competition Compromised;

(d) Owns Suspended Enterprise; and

(e) Suspicious Committee.

The MFrags will be presented from the less dependent to the more dependent. *I.e.*, an MFragment will only be described after all its dependent MFrags have been described. It is consistent with the order of the dependency explanations given previously. Also, all the LPDs defined in this PO are notional only. No real data or statistics was used. Therefore, before this model can be used in production, a few more iterations are necessary in order to make sure these notional probabilities are correct.

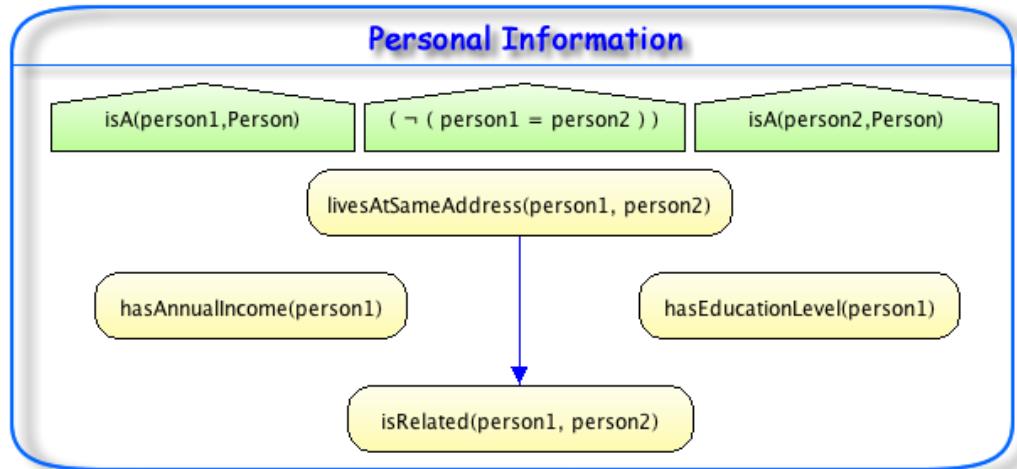


Figure B.1: MFragment Personal Information.

Figure B.1 presents the Personal Information MFragment. In it we have RVs associated to the class `Person`. Listings B.1, B.75, B.3, and B.4 present the LPDs for the RVs, `hasAnnualIncome(person1)`, `hasEducationLevel(person1)`, `livesAtSameAddress(person1, person2)`, and `isRelated(person1, person2)`, respectively. The assumptions behind these LPDs are that a person is more likely to have a lower income, the most people have either middle school or high school education, two random people rarely live at the same address, and if two people live at the same address, they are more likely to be related¹.

Listing B.1: LPD for `hasAnnualIncome(person)`

```

1 [ 
2   Lower10k = .4 ,
3   From10kTo30k = .3 ,
4   From30kTo60k = .2 ,
5   From60kTo100k = .09 ,
6   Greater100k = .01
7 ]

```

Listing B.2: LPD for `hasEducationLevel(person)`

```

1 [ 
2   NoEducation = .1 ,
3   MiddleSchool = .4 ,
4   HighSchool = .3 ,
5   Undergraduate = .15 ,
6   Graduate = .05
7 ]

```

Listing B.3: LPD for `livesAtSameAddress(person1, person2)`

```

1 [ 
2   true = .0001 ,
3   false = .9999
4 ]

```

¹Notice that although it is reasonable to think that the education level of a person influences this person's annual income, in this iteration the PO is not modeled this way. This is not a major issue because we will usually have both information available. However, in future iterations this dependence should be modeled or at least taken into consideration.

Listing B.4: LPD for `isRelated(person1, person2)`

```

1 if any person1.person2 have ( livesAtSameAddress = true ) [
2   true = .9,
3   false = .1
4 ] else [
5   true = .001,
6   false = .999
7 ]

```

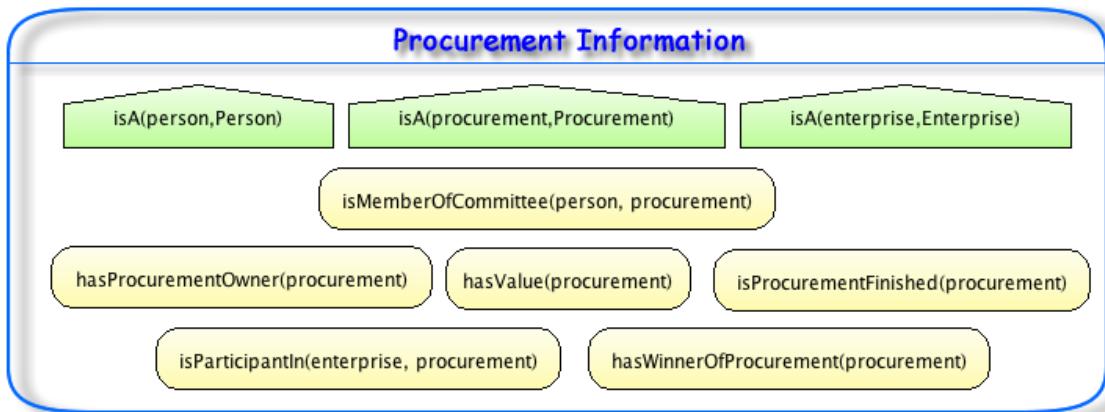


Figure B.2: Mfrag Procurement Information.

Figure B.2 presents the Procurement Information Mfrag. In it we have RVs associated to the class `Procurement`. Listings B.5, B.6, B.7, and B.8 present the LPDs for the RVs, `isMemberOfCommittee(person, procurement)`, `hasValue(procurement)`, `isProcurementFinished(procurement)`, and `isParticipantIn(enterprise, procurement)`, respectively. The assumptions behind these LPDs are that a random person is rarely a member of a committee and a random enterprise is rarely a participant in a procurement. All other RVs assume a uniform distribution, including `hasProcurementOwner(procurement)` and `hasWinnerOfProcurement(procurement)`, which are uniform over all possible instances of `PublicAgency` and `Enterprise`, respectively.

Listing B.5: LPD for `isMemberOfCommittee(person, procurement)`

```
1 [  
2   true = .0001,  
3   false = .9999  
4 ]
```

Listing B.6: LPD for `hasValue(procurement)`

```
1 [  
2   Lower10k = .2,  
3   From10kTo100k = .2,  
4   From100kTo500k = .2,  
5   From500kTo1000k = .2,  
6   Greater1000k = .2  
7 ]
```

Listing B.7: LPD for `isProcurementFinished(procurement)`

```
1 [  
2   true = .5,  
3   false = .5  
4 ]
```

Listing B.8: LPD for `isParticipantIn(enterprise, procurement)`

```
1 [  
2   true = .0001,  
3   false = .9999  
4 ]
```

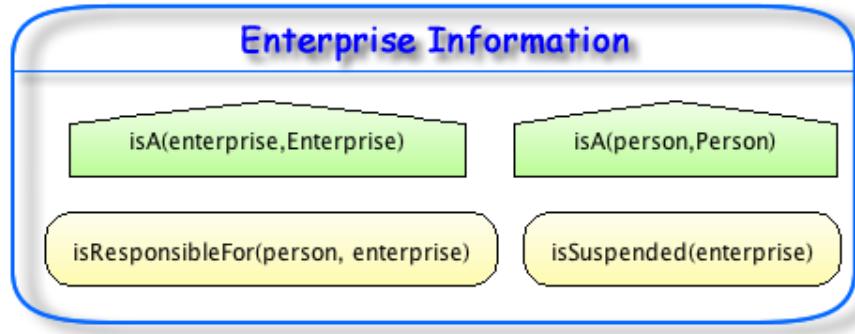


Figure B.3: MFragment Enterprise Information.

Figure B.3 presents the Enterprise Information MFragment. In it we have RVs associated to the class `Enterprise`. Listings B.9 and B.10 present the LPDs for the RVs, `isResponsibleFor(person, enterprise)` and `isSuspended(enterprise)`, respectively. The assumptions behind these LPDs are that a random person is rarely a the owner of an enterprise and a random enterprise is rarely suspended from bidding in public procurements.

Listing B.9: LPD for `isResponsibleFor(person, enterprise)`

```

1 [ 
2   true = .0001,
3   false = .9999
4 ]
```

Listing B.10: LPD for `isSuspended(enterprise)`

```

1 [ 
2   true = .0001,
3   false = .9999
4 ]
```

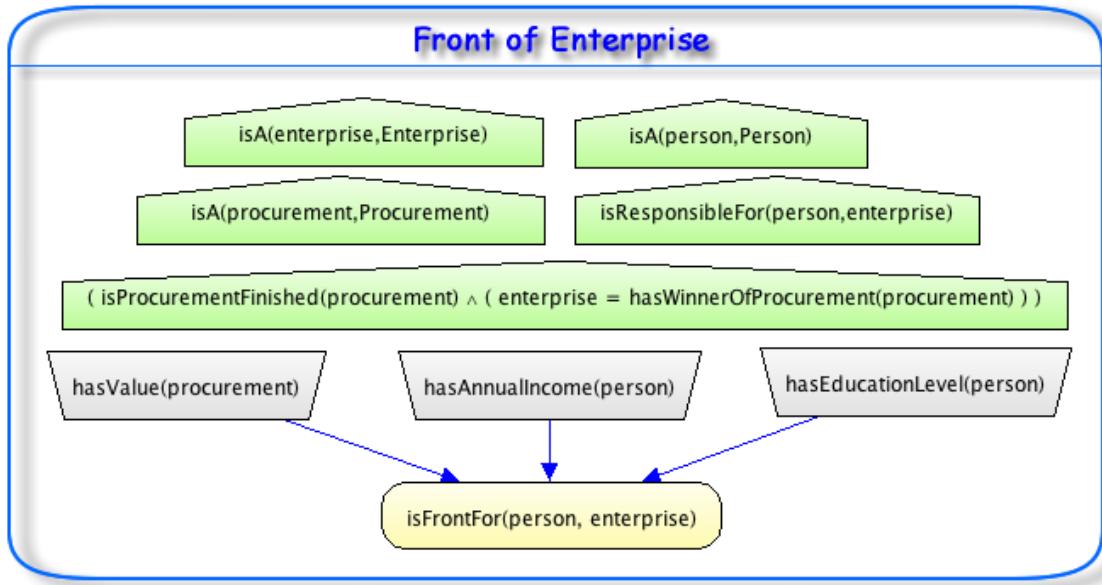


Figure B.4: MFragment Front of Enterprise.

Figure B.4 presents the Front of Enterprise MFragment. Listings B.9 presents the LPD for the RV `isFrontFor(person, enterprise)`. The assumption behind this LPD is that if the enterprise won a procurement of high value, but the owner of the enterprise does not make a lot of money and/or does not have a high education level, then this person is more likely to be a front for this enterprise.

Listing B.11: LPD for `isFrontFor(person, enterprise)`

```

1 if any procurement have ( hasValue = From100kTo500k ) [
2   if any person have ( hasAnnualIncome = Lower10k | hasEducationLevel =
3     NoEducation ) [
4     true = .9,
5     false = .1
6   ] else if any person have ( hasAnnualIncome = From10kTo30k |
7     hasEducationLevel = MiddleSchool ) [
8     true = .6,
9     false = .4
10   ] else [
11     true = .0001,
12     false = .9999
13   ]
14 ] else if any procurement have ( hasValue = From500kTo1000k ) [

```

```

13   if any person have ( hasAnnualIncome = Lower10k | hasEducationLevel =
14     NoEducation ) [
15       true = .95,
16       false = .05
17   ] else if any person have ( hasAnnualIncome = From10kTo30k |
18     hasEducationLevel = MiddleSchool ) [
19       true = .8,
20       false = .2
21   ] else if any person have ( hasAnnualIncome = From30kTo60k |
22     hasEducationLevel = HighSchool ) [
23       true = .6,
24       false = .4
25   ] else [
26       true = .0001,
27       false = .9999
28   ]
29 ] else if any procurement have ( hasValue = Greater1000k ) [
30   if any person have ( hasAnnualIncome = Lower10k | hasEducationLevel =
31     NoEducation ) [
32       true = .99,
33       false = .01
34   ] else if any person have ( hasAnnualIncome = From10kTo30k |
35     hasEducationLevel = MiddleSchool ) [
36       true = .9,
37       false = .1
38   ] else if any person have ( hasAnnualIncome = From30kTo60k |
39     hasEducationLevel = HighSchool ) [
40       true = .8,
41       false = .2
42   ] else if any person have ( hasAnnualIncome = From60kTo100k |
43     hasEducationLevel = Undergraduate ) [
44       true = .6,
45       false = .4
46   ] else [
47       true = .0001,
48       false = .9999
49   ]
50 ]

```

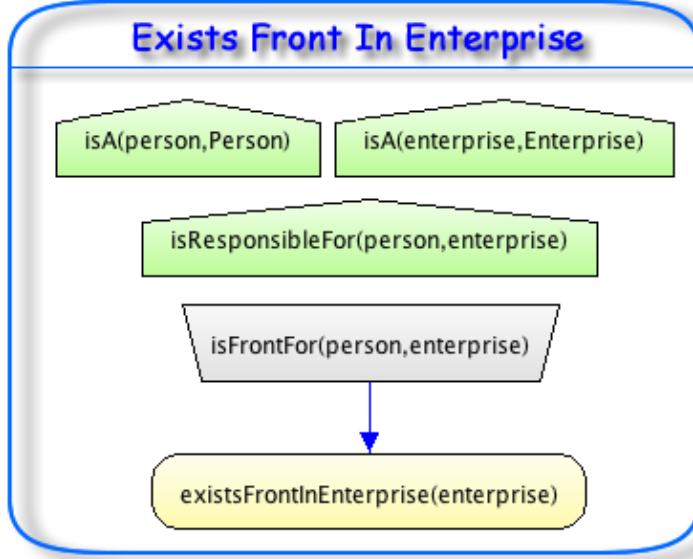


Figure B.5: MFragment Exists Front in Enterprise.

Figure B.5 presents the Exists Front in Enterprise MFragment. Listings B.12 presents the LPD for the RV `existsFrontInEnterprise(enterprise)`. The assumption behind this LPD is that if the enterprise has at least one owner that is a front, then there is a front in this enterprise. Notice this RV represents in fact an existential formula. However, due to limitations in UnBBayes' current version, this existential formula was implemented as a regular RV using the expressiveness of the LPD grammar.

Listing B.12: LPD for `existsFrontInEnterprise(enterprise)`

```

1 if any person.enterprise have ( isFrontFor = true ) [
2   true = 1,
3   false = 0
4 ] else if all person.enterprise have ( isFrontFor = false ) [
5   true = 0,
6   false = 1
7 ] else [
8   true = .0001,
9   false = .9999
10 ]

```

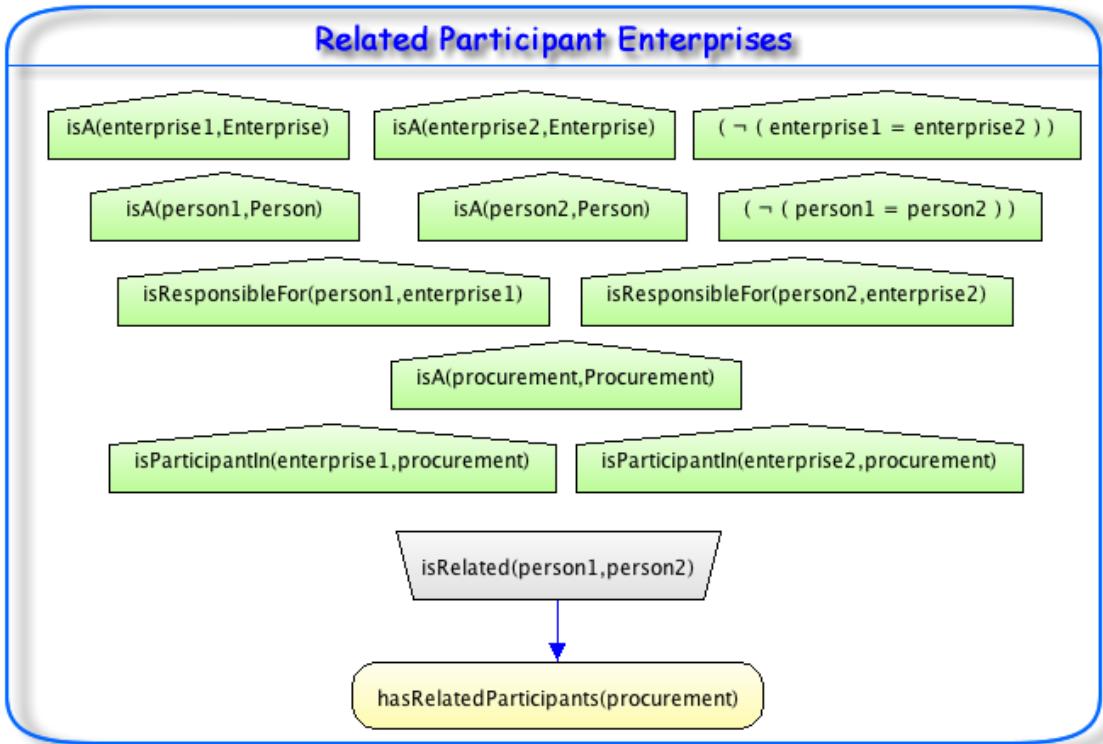


Figure B.6: MFragment Related Participant Enterprises.

Listing B.13: LPD for `hasRelatedParticipants(procurement)`

```

1 if any person1.person2 have ( isRelated = true ) [
2   true = 1,
3   false = 0
4 ] else if all person1.person2 have ( isRelated = false ) [
5   true = 0,
6   false = 1
7 ] else [
8   true = .0001,
9   false = .9999
10 ]

```

Figure B.6 presents the Related Participant Enterprise MFragment. Listings B.13 presents the LPD for the RV `hasRelatedParticipants(procurement)`. The assumption behind this LPD is that if any two enterprises participating in this procurement have owners that

are related, then this procurement has related participants. Notice this RV could also be represented as a formula. However, due to limitations in UnBBayes' current version, this existential formula was implemented as a regular RV using the expressiveness of the LPD grammar.

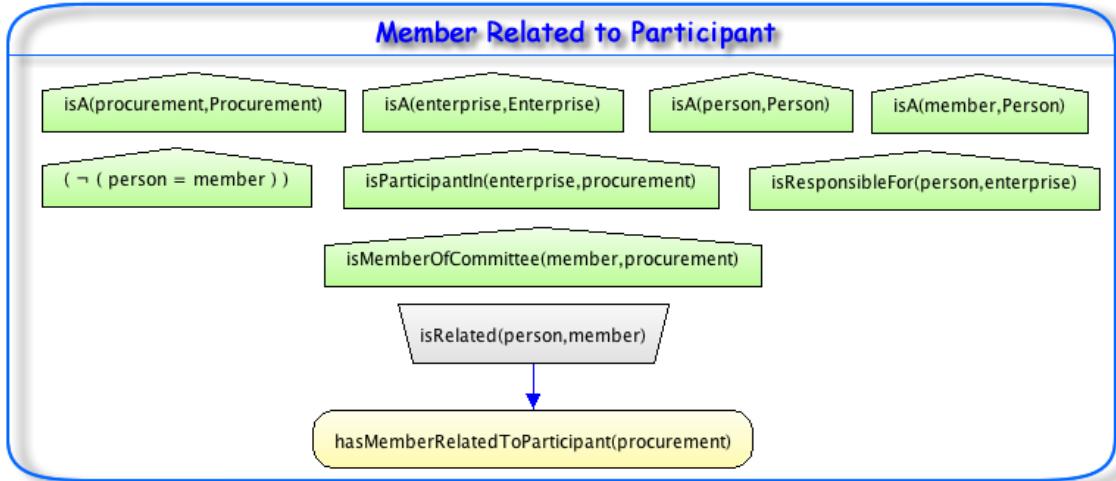


Figure B.7: Mfrag Member Related to Participant.

Listing B.14: LPD for `hasMemberRelatedToParticipant(procurement)`

```

1 if any person.member have ( isRelated = true ) [
2   true = 1,
3   false = 0
4 ] else if all person.member have ( isRelated = false ) [
5   true = 0,
6   false = 1
7 ] else [
8   true = .0001,
9   false = .9999
10 ]

```

Figure B.7 presents the Member Related to Participant Mfrag. Listings B.14 presents the LPD for the RV `hasMemberRelatedToParticipant(procurement)`. The assumption behind this LPD is that if any member of the procurement is related to any owner of any

enterprise participating in this procurement, then this procurement has a member related to a participant. Notice this RV could also be represented as a formula. However, due to limitations in UnBBayes' current version, this existential formula was implemented as a regular RV using the expressiveness of the LPD grammar.

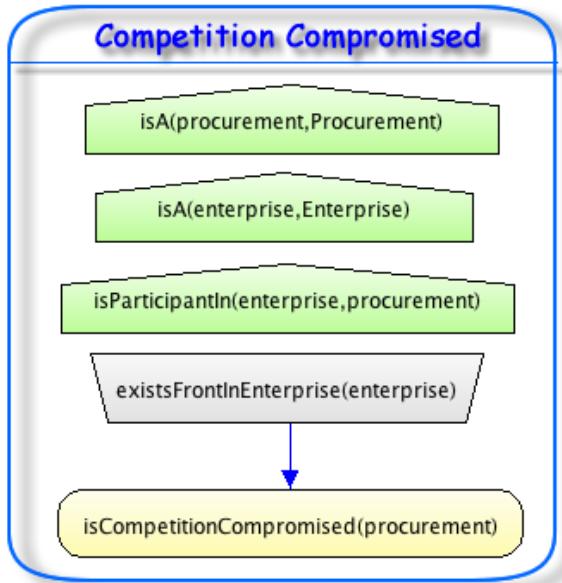


Figure B.8: Mfrag Competition Compromised.

Figure B.8 presents the Competition Compromised Mfrag. Listings B.15 presents the LPD for the RV `isCompetitionCompromised(procurement)`. The assumptions behind this LPD are that: if there exists a front in any of the participating enterprises, or if the participating enterprises are related, or if any member is related to any participating enterprise, then the competition is more likely to be compromised; and that if these things happen together, the probability of having competition compromised is even higher.

Listing B.15: LPD for `isCompetitionCompromised(procurement)`

```

1  if any procurement have ( hasRelatedParticipants = true &
2    hasMemberRelatedToParticipant = true ) [
3    if any enterprise have ( existsFrontInEnterprise = true ) [
4      true = .9,
5      false = .1
6    ] else [
7      true = .8,
8      false = .2
9    ]
10   ] else if any procurement have ( hasRelatedParticipants = true |
11     hasMemberRelatedToParticipant = true ) [
12     if any enterprise have ( existsFrontInEnterprise = true ) [
13       true = .8,
14       false = .2
15     ] else [
16       true = .6,
17       false = .4
18     ]
19   ] else if any enterprise have ( existsFrontInEnterprise = true ) [
20     true = .6,
21     false = .4
22   ] else [
23     true = .0001,
24     false = .9999
25   ]

```

Figure B.9 presents the Owns Suspended Enterprise MFragment. Listings B.16 presents the LPD for the RV `ownsSuspendedEnterprise(person)`. The assumption behind this LPD is that if a person is owner of at least one enterprise suspended from bidding in public procurements, then this person owns a suspended enterprise. Notice this RV could also be represented as a formula. However, due to limitations in UnBBayes' current version, this existential formula was implemented as a regular RV using the expressiveness of the LPD grammar.

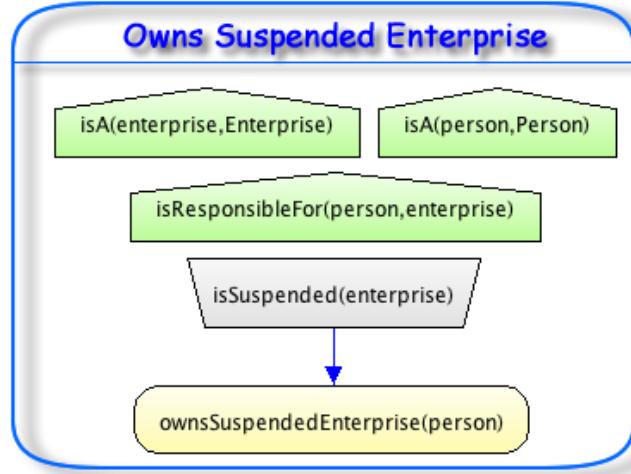


Figure B.9: MFragment Owns Suspended Enterprise.

Listing B.16: LPD for `ownsSuspendedEnterprise(person)`

```

1 if any enterprise have ( isSuspended = true ) [
2   true = 1,
3   false = 0
4 ] else if any enterprise have ( isSuspended = false ) [
5   true = 0,
6   false = 1
7 ] else [
8   true = .001,
9   false = .999
10 ]

```

Figure B.10 presents the Judgment History MFragment. In it we have RVs associated to the judgement (criminal and administrative) history of a Person. Listings B.17, B.18, and B.19 present the LPDs for the RVs, `hasCriminalHistory(person)`, `hasAdministrativeHistory(person)`, and `hasCleanHistory(person)`, respectively. The assumptions behind these LPDs are that a person is more likely to have never been investigated, and the probability of a person having a clean history is lower if he/she was never investigated, higher if he/she was investigated, and extremely high if he/she was convicted².

²Maybe a better name for this node would be `isTrustworthy`. Nevertheless, the idea is that if someone

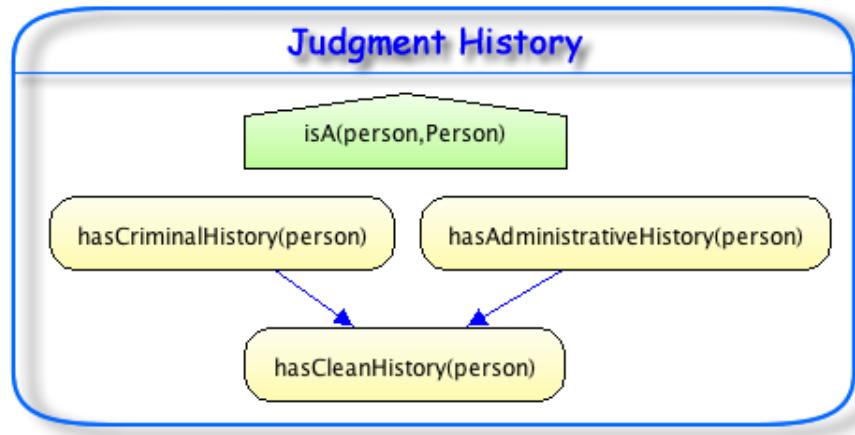


Figure B.10: Mfrag Judgment History.

Listing B.17: LPD for hasCriminalHistory(person)

```

1 [ 
2   Convicted = .0001,
3   Investigated = .001,
4   NeverInvestigated = .9989
5 ]

```

Listing B.18: LPD for hasAdministrativeHistory(person)

```

1 [ 
2   Convicted = .0001,
3   Investigated = .001,
4   NeverInvestigated = .9989
5 ]

```

was investigated and/or convicted then he might not be a good candidate for being part of a procurement committee.

Listing B.19: LPD for hasCleanHistory(person)

```

1 if any person have ( hasCriminalHistory = Convicted | hasAdministrativeHistory
2   = Convicted ) [
3   true = .01,
4   false = .99
5 ] else if any person have ( hasCriminalHistory = Investigated | 
6   hasAdministrativeHistory = Investigated ) [
7   true = .60,
8   false = .40
9 ] else [
10  true = .99,
11  false = .01
12 ]

```

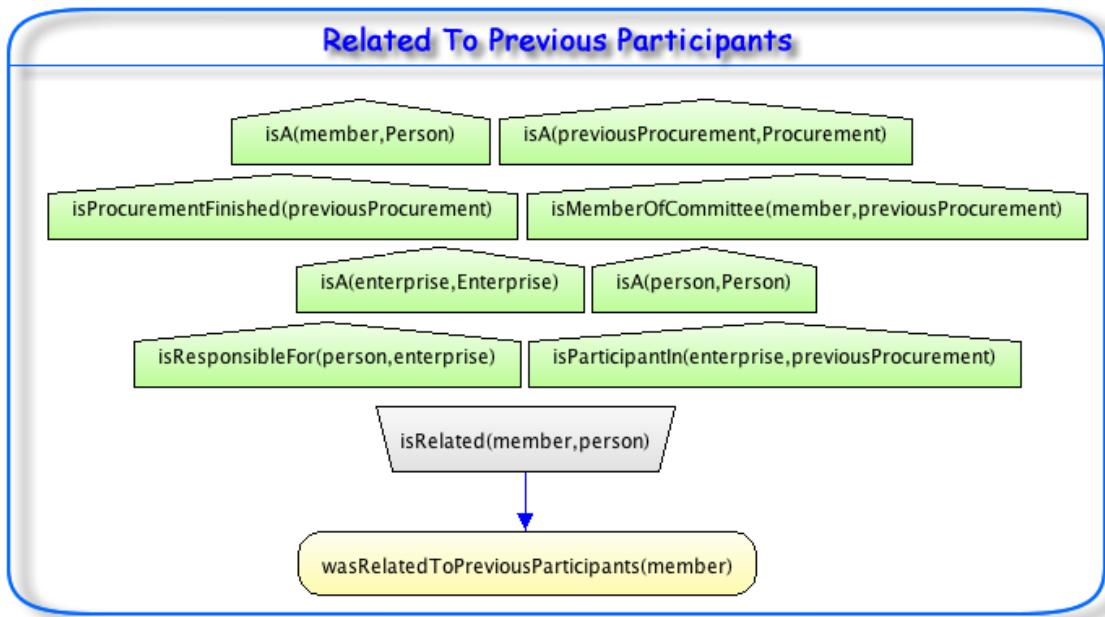


Figure B.11: Mfrag Related to Previous Participants.

Figure B.11 presents the Related to Previous Participants Mfrag. Listings B.20 presents the LPD for the RV `wasRelatedToPreviousParticipants(member)`. The assumption behind this LPD is that if a person was related to any owner of any enterprise participating in any previous procurement (procurement that is finished), when this person was a member

of that procurement, then this member was related to previous participants. Notice this RV could also be represented as a formula. However, due to limitations in UnBBayes' current version, this existential formula was implemented as a regular RV using the expressiveness of the LPD grammar.

Listing B.20: LPD for `wasRelatedToPreviousParticipants(member)`

```

1 if any member.person have ( isRelated = true ) [
2   true = 1,
3   false = 0
4 ] else if all member.person have ( isRelated = false ) [
5   true = 0,
6   false = 1
7 ] else [
8   true = .0001,
9   false = .9999
10 ]

```

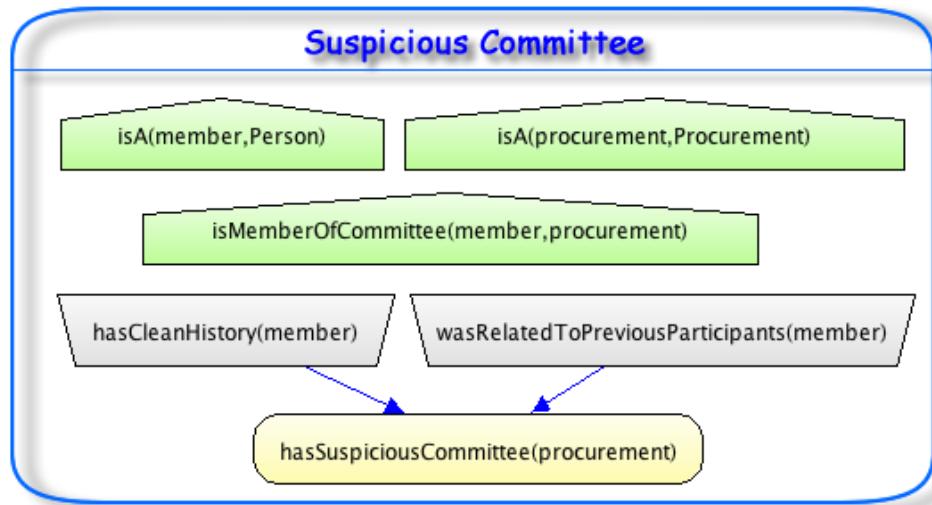


Figure B.12: Mfrag Suspicious Committee.

Figure B.12 presents the Suspicious Committee Mfrag. Listings B.21 presents the LPD for the RV `hasSuspiciousCommittee(procurement)`. The assumptions behind this LPD

are that: if any committee member of this procurement does not have a clean history, or if any committee member was related to previous participants, then the committee is more likely to be suspicious; and that if these things happen together, the probability of having suspicious committee is even higher.

Listing B.21: LPD for `hasSuspiciousCommittee(procurement)`

```

1  if any member have ( wasRelatedToPreviousParticipants = true ) [
2    if any member have ( hasCleanHistory = false ) [
3      true = .9,
4      false = .1
5    ] else [
6      true = .7,
7      false = .3
8    ]
9  ] else if any member have ( wasRelatedToPreviousParticipants = false ) [
10   if any member have ( hasCleanHistory = false ) [
11     true = .7,
12     false = .3
13   ] else [
14     true = .001,
15     false = .999
16   ]
17 ] else [
18   true = .001,
19   false = .999
20 ]

```

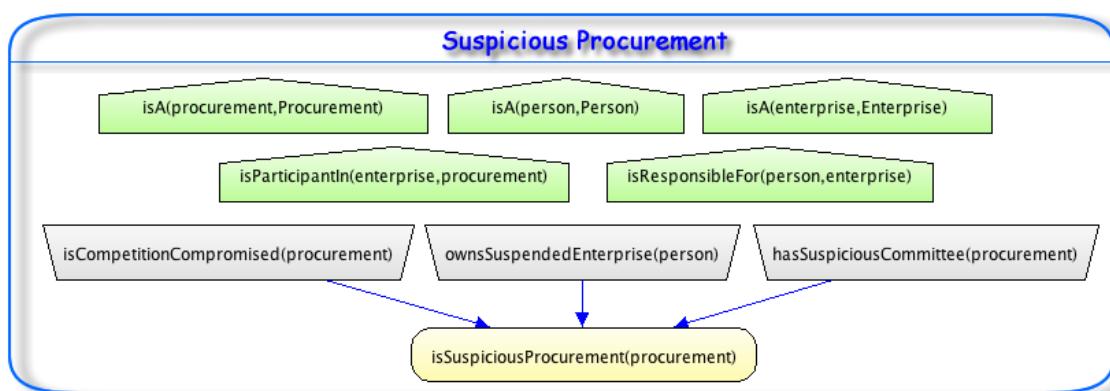


Figure B.13: Mfrag Suspicious Procurement.

Figure B.13 presents the Suspicious Procurement MFragment. Listings B.22 presents the LPD for the RV `isSuspiciousProcurement(procurement)`. The assumptions behind this LPD are that: if the competition is compromised, or if any owner of a participating enterprise owns a suspended enterprise, or if committee of this procurement is suspicious, then the procurement is more likely to be suspicious; and that if these things happen together, the probability of having suspicious procurement is even higher.

Listing B.22: LPD for `isSuspiciousProcurement(procurement)`

```

1  if any procurement have ( isCompetitionCompromised = true &
2    hasSuspiciousCommittee = true ) [
3    if any person have ( ownsSuspendedEnterprise = true ) [
4      true = .90,
5      false = .10
6    ] else [
7      true = .80,
8      false = .20
9    ]
10   ] else if any procurement have ( isCompetitionCompromised = true |
11     hasSuspiciousCommittee = true ) [
12     if any person have ( ownsSuspendedEnterprise = true ) [
13       true = .80,
14       false = .20
15     ] else [
16       true = .70,
17       false = .30
18     ]
19   ] else [
20     if any person have ( ownsSuspendedEnterprise = true ) [
21       true = .70,
22       false = .30
23     ] else [
24       true = .0001,
25       false = .9999
26     ]
27   ]
28 ]
29 
```

B.2 Probabilistic Ontology for Maritime Domain Awareness

All the assumptions for the RVs created and for defining their LPD will be described for every MFragment designed for the MTheory that represents the PO for the MDA implemented. In each MFragment, the resident RVs are shown as yellow rounded rectangles; the input RVs are

shown as gray trapezoids; the context RVs are shown as green pentagons.

B.2.1 Fist Iteration

In order to make reference easier, the rules defined during Analysis & Design in Chapter 5 Subsection 5.2.1 will be repeated here. The rules are:

1. A ship is of interest if and only if it has a terrorist crew member;
2. If a crew member is related to a terrorist, then it is more likely that he is also a terrorist;
3. If a crew member is a member of a terrorist organization, then it is more likely that he is a terrorist;
4. If an organization has a terrorist member, it is more likely that it is a terrorist organization;
5. A ship of interest is more likely to have an unusual route;
6. A ship of interest is more likely to meet other ships for trading illicit cargo;
7. A ship that meets other ships to trade illicit cargo is more likely to have an unusual route;
8. A ship of interest is more likely to have an evasive behavior;
9. A ship with evasive behavior is more likely to have non responsive electronic equipment;
10. A ship with evasive behavior is more likely to deploy an ECM;
11. A ship might have non responsive electronic equipment due to working problems;
12. A ship that is within radar range of a ship that deployed an ECM might be able to detect the ECM, but not who deployed it.

The primary goal is shown in the Ship of Interest MFragment in Figure B.14. This MFragment has only one resident node, `isShipOfInterest(ship)`. The only context node present in this MFragment define the type for the variable `ship`, which is the `Ship` entity. The input node `hasTerroristCrew(ship)` is defined in another MFragment, which will be explained later.

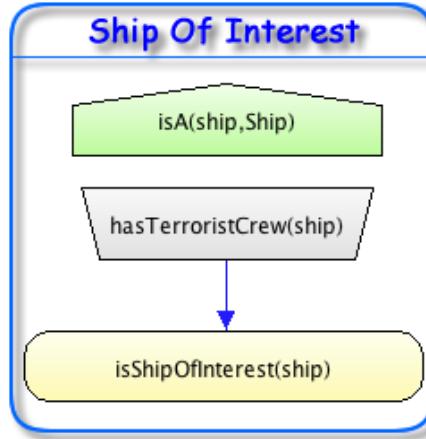


Figure B.14: MFragment for identifying the ship of interest.

The LPD for the resident node `isShipOfInterest(ship)` follows rule 1, *i.e.*, if the ship has a terrorist crew, then it is a ship of interest for sure, otherwise, it is unlikely, which was considered as 0.1%. See Listing B.23 for the complete LPD.

Listing B.23: LPD for `isShipOfInterest(ship)`

```

1  if any ship have ( hasTerroristCrew = true ) [
2      true = 1,
3      false = 0
4  ] else [
5      true = .001,
6      false = .999
7 ]

```

The question related to the identification of a terrorist crew member is presented in the Has Terrorist Crew, Terrorist Person, and Ship Characteristics MFrags in Figure B.15. The

context nodes on all these MFrags refer only to the types of the variables, where `person`, `ship`, and `org`, refer to `Person`, `Ship`, and `Organization`, respectively.

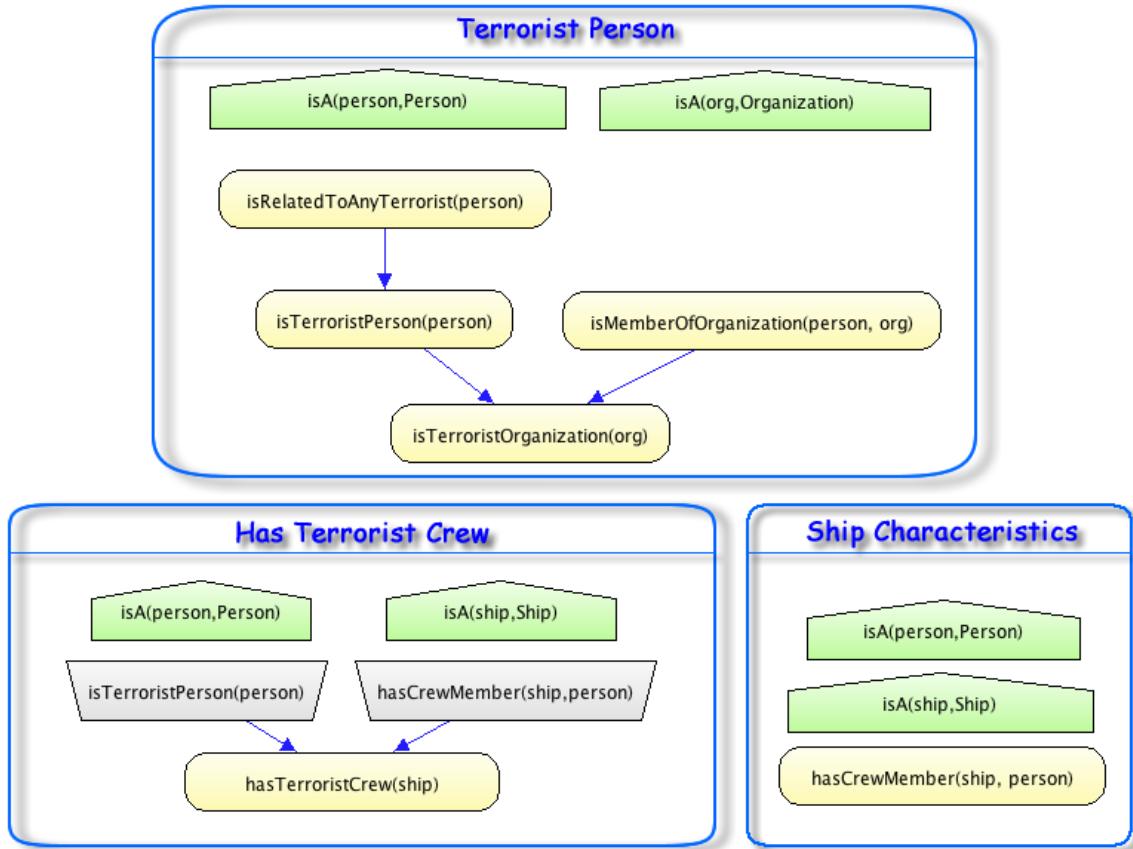


Figure B.15: MFrags for identifying a terrorist crew member.

The LPD for the resident node `isCrewMember(person, ship)` has a prior probability of 0.5% of being true, *i.e.*, 5 out of 1000 people are crew members of a given ship. In reality, given all people in the world, this ratio might be much smaller, however, we assume that we are dealing with a subset of people that is more likely to be crew members. See Listing B.24 for the complete LPD.

Listing B.24: LPD for `isCrewMember(person, ship)`

```

1  [
2    true = .005,
3    false = .995
4 ]

```

The LPD for the resident node `hasTerroristCrew(ship)` follows rule 1. Here we just have a logical statement saying that the ship has a terrorist crew if and only if a person is a terrorist and also a crew member of this ship. If no information about its parents is known, the default distribution is used. In this case, we assume that it is unlikely that a ship has a terrorist crew, which is interpreted as 0.1%. See Listing B.25 for the complete LPD.

Listing B.25: LPD for `hasTerroristCrew(ship)`

```

1  if any person have ( isTerroristPerson = true ) [
2    if any person.ship have ( isCrewMember = true ) [
3      true = 1,
4      false = 0
5    ] else [
6      true = 0,
7      false = 1
8    ]
9  ] else if any person have ( isTerroristPerson = false ) [
10   if any person.ship have ( isCrewMember = true ) [
11     true = 0,
12     false = 1
13   ] else [
14     true = 0,
15     false = 1
16   ]
17 ] else [
18   true = .001,
19   false = .999
20 ]

```

The LPD for the resident node `isRelatedToAnyTerrorist(person)` has a prior probability of 0.1% of being true, which means that a person is unlikely to be related to a terrorist. This information is provided by a social network system by looking at the relation `isRelatedTo` and classes `Person` and `Terrorist` presented on our design. If there is one `Terrorist`

who is related to a person, then `isRelatedToAnyTerrorist(person)` is true. So we simplified our PO by just representing the relation `isRelatedToAnyTerrorist(person)`. See Listing B.26 for the complete LPD.

Listing B.26: LPD for `isRelatedToAnyTerrorist(person)`

```

1 [
2   true = .001,
3   false = .999
4 ]
```

The LPD for the resident node `isTerroristPerson(person)` follows rule 2. If person is related to any terrorist, then this person is more likely to be a terrorist. Otherwise, it is unlikely that this person is a terrorist. Here more likely is interpreted as 70% and unlikely as 0.1%. See Listing B.58 for the complete LPD.

Listing B.27: LPD for `isTerroristPerson(person)`

```

1 if any person have ( isRelatedToAnyTerrorist = true ) [
2   true = .7,
3   false = .3
4 ] else [
5   true = .001,
6   false = .999
7 ]
```

The LPD for the resident node `isMemberOfOrganization(person, org)` has a prior probability of 1%, which means that one in every one hundred people is a member of a given organization. Again, this might be a much smaller ratio in reality, but here we assume we are dealing with a subset of people that are more likely to be members of a given organization. See Listing B.28 for the complete LPD.

Listing B.28: LPD for `isMemberOfOrganization(person, org)`

```

1 [
2   true = .01,
3   false = .99
4 ]
```

The LPD for the resident node `isTerroristOrganization(org)` follows rules 3 and 4. If there is a person that is a terrorist and is also a member of a given organization, then this organization is likely to be a terrorist organization, otherwise, it is unlikely to be a terrorist organization. Here, we assume likely to be 90% and unlikely to be 0.1%. See Listing B.29 for the complete LPD.

Listing B.29: LPD for `isTerroristOrganization(org)`

```

1  if any person have ( isTerroristPerson = true ) [
2    if any person.org have ( isMemberOfOrganization = true ) [
3      true = .9,
4      false = .1
5    ] else [
6      true = .001,
7      false = .999
8    ]
9  ] else [
10   true = .001,
11   false = .999
12 ]

```

The question related to the identification of unusual routes is presented on the Unusual Route and Meeting MFrags in Figure B.16. In both MFrags there are two context nodes to define the types of the variables `ship1` and `ship2`, which is entity `Ship`. Besides that, there is one context node that defined that `ship1` has to be different than `ship2`.

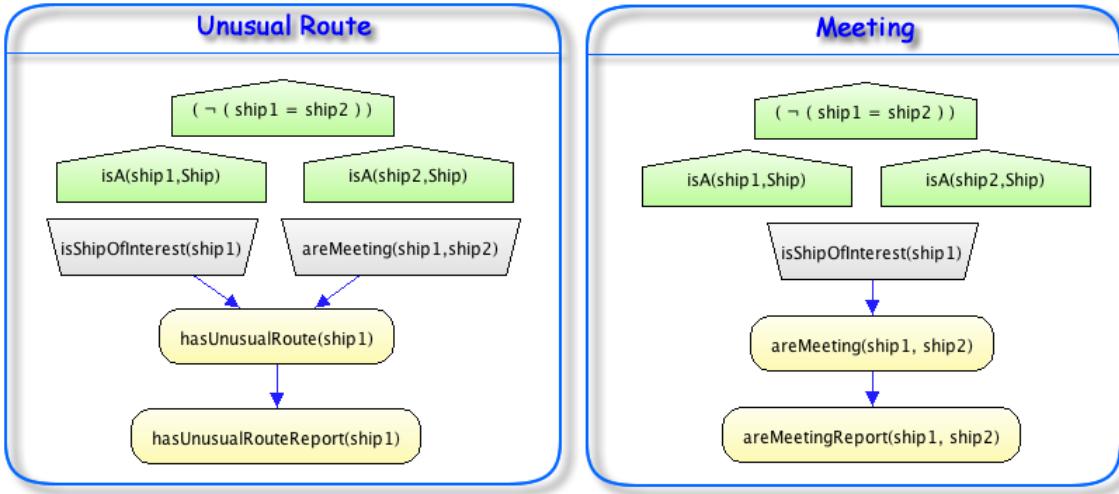


Figure B.16: MFrags for identifying the ship with unusual route.

The LPD for the resident node `areMeeting(ship1, ship2)` follows rule 6. If ship is a ship of interest, then it is more likely to meet other ships for trading illicit cargo. Otherwise, it is unlikely that this ship will meet other ships. In this case, more likely is interpreted as 75% and unlikely as 0.1%. See Listing B.30 for the complete LPD.

Listing B.30: LPD for `areMeeting(ship1, ship2)`

```

1 if any ship1 have ( isShipOfInterest = true ) [
2   true = .75,
3   false = .25
4 ] else [
5   true = .001,
6   false = .999
7 ]

```

The LPD for the resident node `areMeetingReport(ship1, ship2)` does not follow a specific rule, however, it is associated to the fact that receiving a report about an event is not the same as the event itself (see [119] for more details), *i.e.* even though someone states that two ships met, it might be the case that whoever gave the report was mistaken. These issues are not addressed in detail in this project, but we have assumed that when

two ships meet, there is a 90% chance that the report will say these two ships met, and if two ships have not met, there is a 80% chance that the report will say these two ships have not met. See Listing B.31 for the complete LPD.

Listing B.31: LPD for `areMeetingReport(ship1, ship2)`

```

1 if any ship1.ship2 have ( areMeeting = true ) [
2   true = .9,
3   false = .1
4 ] else [
5   true = .2,
6   false = .8
7 ]

```

The LPD for the resident node `hasUnusualRoute(ship1)` follows rules 5 and 7. If ship is of interest and is meeting other ships, then it is more likely the ship has an unusual route. However, if ship is of interest but is not meeting other ships, then it is likely (but less than the previous case) the ship has an unusual route. If ship is not of interest, then it does not matter if it is meeting other ships. In this scenario, this ship is unlikely to be using an unusual route. Here it is assumed more likely as 90%, likely as 75%, and unlikely as 0.1%. See Listing B.32 for the complete LPD.

Listing B.32: LPD for `hasUnusualRoute(ship1)`

```

1 if any ship1 have ( isShipOfInterest = true ) [
2   if any ship1.ship2 have ( areMeeting = true ) [
3     true = .9,
4     false = .1
5   ] else [
6     true = .75,
7     false = .25
8   ]
9 ] else [
10   true = .001,
11   false = .999
12 ]

```

The LPD for the resident node `hasUnusualRouteReport(ship1)` does not follow a specific rule, however, it is associated to the fact that receiving a report about an event

is not the same as the event itself, as in the case of `areMeetingReport(ship1, ship2)`. It is assumed that when the ship has an unusual route, there is a 90% chance that the report will say the ship has an unusual route, and if the ship has a normal route, there is an 80% chance that the report will say the ship has a normal route. See Listing B.33 for the complete LPD.

Listing B.33: LPD for `hasUnusualRouteReport(ship1)`

```

1 if any ship1 have ( hasUnusualRoute = true ) [
2   true = .9 ,
3   false = .1
4 ] else [
5   true = .2 ,
6   false = .8
7 ]

```

The question related to identification of evasive behavior is shown in the Evasive Behavior, Electronics Status, and Radar MFrags in Figure B.17. As in the previous MFrags, the variables `ship`, `ship1`, and `ship2` have their type defined by their context node as entity `Ship`. In Radar and Evasive Behavior MFrags, `ship1` and `ship2` are defined as two different ships. Finally, the relations in Evasive Behavior Mfrag are only valid when `ship1` is within the radar range of `ship2`.

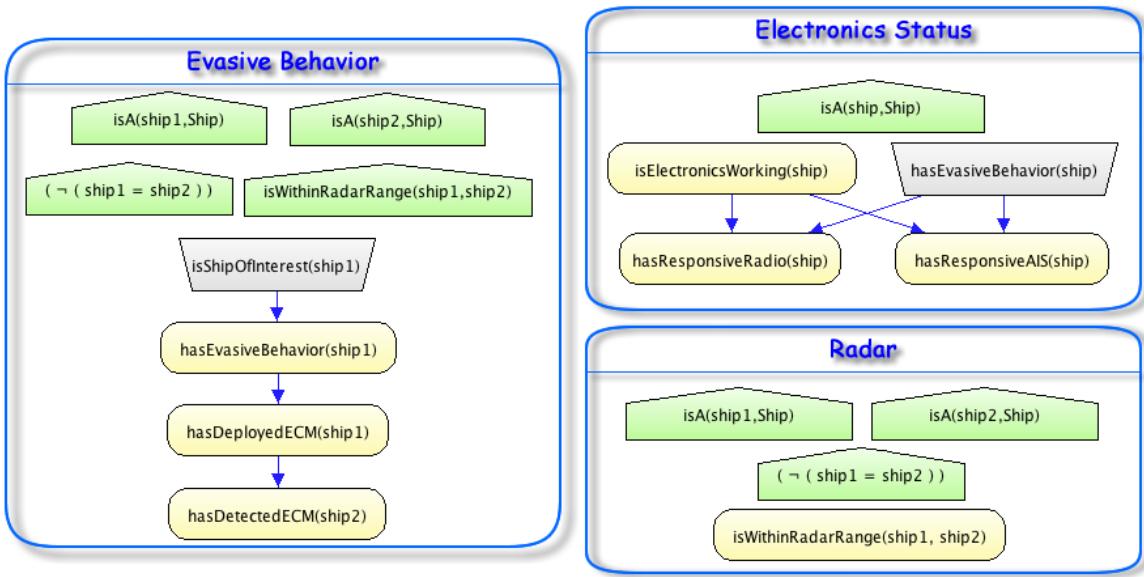


Figure B.17: MFrags for identifying the ship with evasive behavior.

The LPD for the resident node `isWithinRadarRange(ship1, ship2)` has a prior of 0.5% of having `ship1` in range of `ship2`s radar. Again, this is just a subjective analysis and it is not intended to represent a real frequency. This is also a simplification from the design we had. Some external system is going to compare the position of `ship1` with the position of `ship2` and verify if this distance is smaller or equal to `ship2`s radar range. See Listing B.34 for the complete LPD.

Listing B.34: LPD for `isWithinRadarRange(ship1, ship2)`

```

1  [
2      true  = .005 ,
3      false = .995
4  ]

```

The LPD for the resident node `hasEvasiveBehavior(ship1)` follows rule 8. If `ship1` is of interest, then it is more likely to have an evasive behavior. Otherwise it is unlikely to have an evasive behavior. It is assumed more likely as 75% and unlikely as 0.1% in this

case. See Listing B.35 for the complete LPD.

Listing B.35: LPD for `hasEvasiveBehavior(ship1)`

```
1 if any ship1 have ( isShipOfInterest = true ) [
2   true = .75,
3   false = .25
4 ] else [
5   true = .001,
6   false = .999
7 ]
```

The LPD for the resident node `hasDeployedECM(ship1)` follows rule 10. If `ship1` has evasive behavior, then it is more likely to deploy an ECM. Otherwise, it is unlikely to deploy an ECM. It is assumed more likely as 75% and unlikely as 0.1% in this case. See Listing B.36 for the complete LPD.

Listing B.36: LPD for `hasDeployedECM(ship1)`

```
1 if any ship1 have ( hasEvasiveBehavior = true ) [
2   true = .75,
3   false = .25
4 ] else [
5   true = .001,
6   false = .999
7 ]
```

The LPD for the resident node `hasDetectedECM(ship2)` follows rule 12. If a `ship1` that deployed an ECM is within radar range of `ship2`, then it is likely that `ship2` will detect ECM, but not who deployed it. Otherwise, it is unlikely that `ship2` will detect an ECM. It is assumed likely as 90% and unlikely as 0.1% in this case. See Listing B.37 for the complete LPD.

Listing B.37: LPD for `hasDetectedECM(ship2)`

```
1 if any ship1 have ( hasDeployedECM = true ) [  
2   true = .9,  
3   false = .1  
4 ] else [  
5   true = .001,  
6   false = .999  
7 ]
```

The LPD for the resident node `isElectronicsWorking(ship)` has a prior of 95% of being working, which means that it is likely that the electronics in a ship is working. Here we simplified our model by grouping all electronics equipment and saying that if one is not working, then this RV should be `false`, otherwise, it is `true`. This is different than our design, which has a property `isWorking` for every electronic. See Listing B.38 for the complete LPD.

Listing B.38: LPD for `isElectronicsWorking(ship)`

```
1 [  
2   true = .95,  
3   false = .05  
4 ]
```

The LPD for the resident node `hasResponsiveRadio(ship)` follows rules 9 and 11. If `ship` has an evasive behavior and the electronics is not working than it is very likely to have non-responsive radio. However, if `ship` has an evasive behavior or the electronics is not working, but not both, then it is likely to have non-responsive radio. In any other case, it is very likely to have responsive radio. It is assumed very likely as 99% and likely as 90%, in this case. See Listing B.39 for the complete LPD.

Listing B.39: LPD for `hasResponsiveRadio(ship)`

```

1 if any ship have ( hasEvasiveBehavior = true ) [
2   if any ship have ( isElectronicsWorking = false ) [
3     true = .01,
4     false = .99
5   ] else [
6     true = .1,
7     false = .9
8   ]
9 ] else if any ship have ( hasEvasiveBehavior = false ) [
10  if any ship have ( isElectronicsWorking = false ) [
11    true = .1,
12    false = .9
13  ] else [
14    true = .99,
15    false = .01
16  ]
17 ] else [
18  true = .99,
19  false = .01
20 ]

```

The LPD for the resident node `hasResponsiveAIS(ship)` follows rules 9 and 11. If `ship` has an evasive behavior and the electronics is not working than it is very likely to have non-responsive AIS. However, if `ship` has an evasive behavior or the electronics is not working, but not both, then it is likely to have non-responsive AIS. In any other case, it is very likely to have responsive AIS. It is assumed very likely as 99% and likely as 90%, in this case. See Listing B.40 for the complete LPD.

Listing B.40: LPD for `hasResponsiveAIS(ship)`

```

1 if any ship have ( hasEvasiveBehavior = true ) [
2   if any ship have ( isElectronicsWorking = false ) [
3     true = .01,
4     false = .99
5   ] else [
6     true = .1,
7     false = .9
8   ]
9 ] else if any ship have ( hasEvasiveBehavior = false ) [
10  if any ship have ( isElectronicsWorking = false ) [
11    true = .1,
12    false = .9
13  ] else [
14    true = .99,
15    false = .01
16  ]
17 ] else [
18  true = .99,
19  false = .01
20 ]

```

B.2.2 Second Iteration

In order to make reference easier, the rules defined during Analysis & Design in Chapter 5 Subsection 5.2.2 will be repeated here. The rules are:

1. *A ship is of interest if and only if it has a terrorist crew member plan;*
2. A ship has a terrorist plan if and only if it has terrorist crew member or if it was hijacked;
3. *If a crew member is related to a terrorist, then it is more likely that he is also a terrorist;*
4. *If a crew member is a member of a terrorist organization, then it is more likely that he is a terrorist;*
5. *If an organization has a terrorist member, it is more likely that it is a terrorist organization;*

6. *A ship of interest is more likely to have an unusual route*, independent of its intention;
7. *A ship of interest, with plans of exchanging illicit cargo, is more likely to meet other ships*;
8. *A ship that meets other ships to trade illicit cargo is more likely to have an unusual route*;
9. A fishing ship is more likely to have a normal change in its destination (*e.g.*, to sell the fish caught) than merchant ships;
10. A normal change in destination will probably change the usual route of the ship;
11. *A ship of interest, with plans of exchanging illicit cargo, is more likely to have an evasive behavior*;
12. A ship with evasive behavior is more likely to have non responsive electronic equipment;
13. A ship might have non responsive electronic equipment due to maintenance problems;
14. ~~A ship with evasive behavior is more likely to deploy an ECM~~;
15. ~~A ship that is within radar range of a ship that deployed an ECM might be able to detect the ECM, but not who deployed it~~;
16. A ship of interest, with plans of exchanging illicit cargo, is more likely to have an erratic behavior;
17. A ship with normal behavior usually does not have the crew visible on the deck;
18. A ship with erratic behavior usually has the crew visible on the deck;
19. If the ship has some equipment failure, it is more likely to see the crew on the deck in order to fix the problem;

20. A ship of interest, independent of its intention, is more likely to have an aggressive behavior;
21. A ship with aggressive behavior is more likely to have weapons visible and to jettison cargo;
22. A ship with normal behavior is not likely to have weapons visible nor to jettison cargo.

Rules inherited from the first iteration are in italic. Items crossed out refer to rules that were considered in the first iteration, but now they have been changed or removed.

This Section will only describe the MFrags and LPDs that have been changed or added in the second iteration. Please refer to Section B.2.1 for those that remain the same. Figure B.18 presents the MTheory created in the second iteration with information on which MFrags are the same, which are new, and which were changed.

Figure B.19 presents the Ship Characteristics MFragment. It adds the RVs `hasTypeOfShip(ship)` and `isHijacked(ship)`, which have their LPDs described by Listings B.41 and B.42, respectively. The assumptions behind these LPDs are that a ship is slightly more likely to be a fishing ship than a merchant ship and a ship is unlikely to be hijacked.

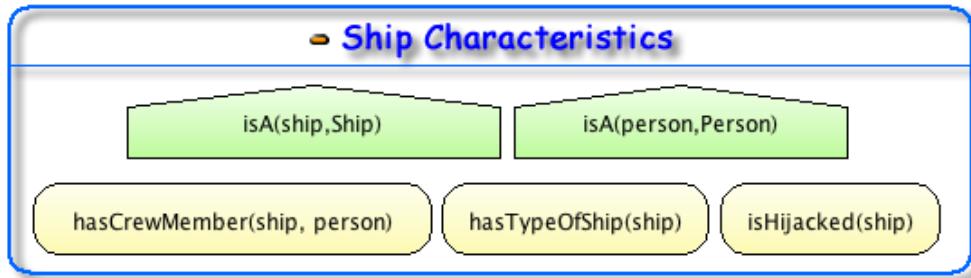


Figure B.19: Ship Characteristics MFragment.

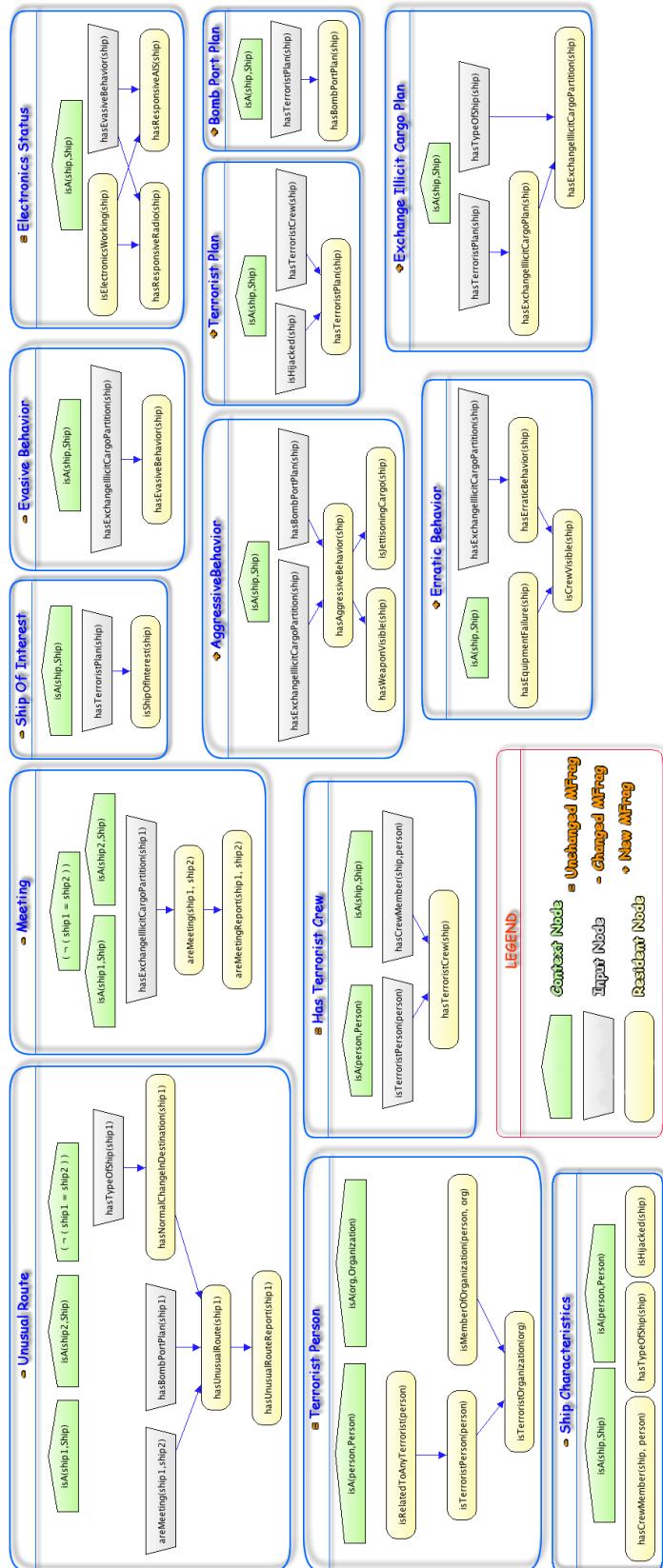


Figure B.18: Mtheory created in second iteration.

Listing B.41: LPD for `hasTypeOfShip(ship)`

```
1 [  
2   Fishing = .6 ,  
3   Merchant = .4  
4 ]
```

Listing B.42: LPD for `isHijacked(ship)`

```
1 [  
2   true = .05 ,  
3   false = .95  
4 ]
```

Figure B.20 presents the Terrorist Plan Mfrag. Listing B.43 presents the LPD for the RV `hasTerroristPlan(ship)`. The assumption behind this LPD is that a ship has a terrorist plan if and only if it has terrorist crew member or if it was hijacked (rule 2).

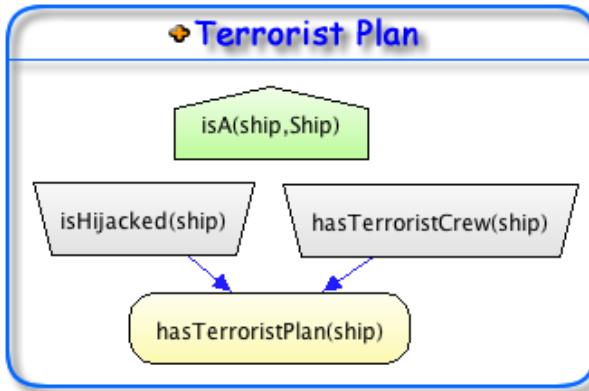


Figure B.20: Terrorist Plan Mfrag.

Listing B.43: LPD for `hasTerroristPlan(ship)`

```

1 if any ship have ( hasTerroristCrew = true | isHijacked = true ) [
2   ExchangeIllicitCargoPlan = .7,
3   BombPortPlan = .3,
4   NoPlan = 0
5 ] else [
6   ExchangeIllicitCargoPlan = 0,
7   BombPortPlan = 0,
8   NoPlan = 1
9 ]

```

Figure B.21 presents the Bomb Port Plan Mfrag. Listing B.44 presents the LPD for the RV `hasBombPortPlan(ship)`. Here we just have a deterministic rule saying that if `ship` has the terrorist plan `BombPortPlan`, then the RV `hasBombPortPlan(ship)` is true.

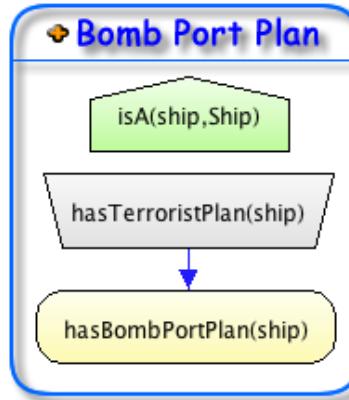


Figure B.21: Bomb Port Plan Mfrag.

Listing B.44: LPD for `hasBombPortPlan(ship)`

```

1 if any ship have ( hasTerroristPlan = BombPortPlan ) [
2   true = 1,
3   false = 0
4 ] else [
5   true = 0,
6   false = 1
7 ]

```

Figure B.22 presents the Exchange Illicit Cargo Plan MFragment. Listings B.45 and B.46 present the LPDs for the RVs `hasBombPortPlan(ship)` and `hasExchangeIllicitCargoPartition(ship)`, respectively. Here we just have a deterministic rule saying that if `ship` has the terrorist plan `ExchangeIllicitCargoPlan`, then the RV `hasExchangeIllicitCargoPlan(ship)` is true. The other node defines a exchange illicit cargo partition based on the type of the ship.

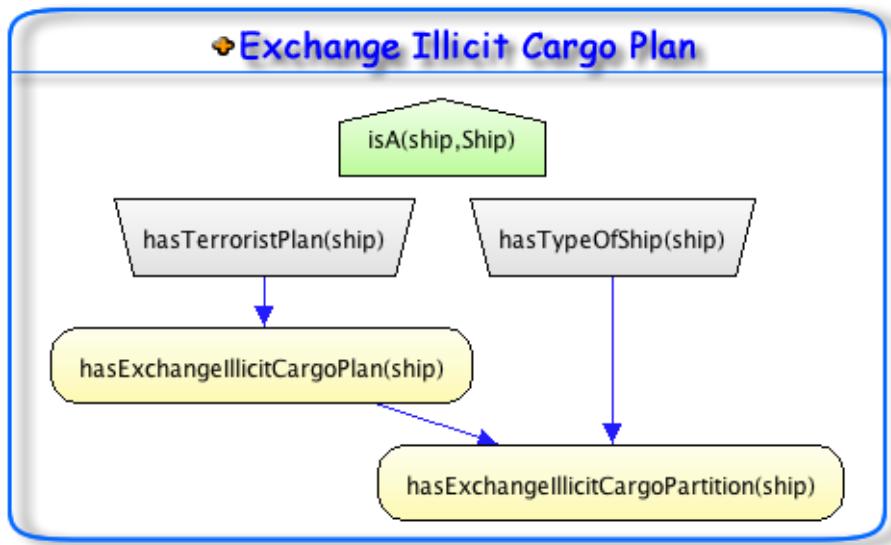


Figure B.22: Exchange Illicit Cargo Plan MFragment.

Listing B.45: LPD for `hasExchangeIllicitCargoPlan(ship)`

```

1  if any ship have ( hasTerroristPlan = ExchangeIllicitCargoPlan ) [
2    true = 1,
3    false = 0
4  ] else [
5    true = 0,
6    false = 1
7 ]
  
```

Listing B.46: LPD for `hasExchangeIllicitCargoPartition(ship)`

```

1 if any ship have ( hasExchangeIllicitCargoPlan = true ) [
2   if any ship have ( hasTypeOfShip = Fishing ) [
3     ExchangeIllicitCargoPlanForFishingShip = 1,
4     ExchangeIllicitCargoPlanForMerchantShip = 0,
5     NoPlan = 0
6   ] else [
7     ExchangeIllicitCargoPlanForFishingShip = 0,
8     ExchangeIllicitCargoPlanForMerchantShip = 1,
9     NoPlan = 0
10 ]
11 ] else [
12   ExchangeIllicitCargoPlanForFishingShip = 0,
13   ExchangeIllicitCargoPlanForMerchantShip = 0,
14   NoPlan = 1
15 ]

```

Figure B.23 presents the Ship of Interest Mfrag. Listing B.47 presents the LPD for the RV `isShipOfInterest(ship)`. The assumption behind this LPD is that a ship is of interest if and only if it has a terrorist plan (rule 1).

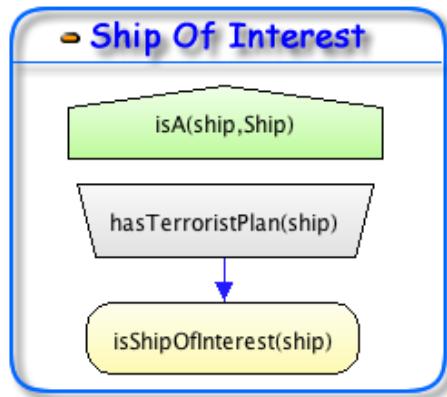


Figure B.23: Ship of Interest Mfrag.

Listing B.47: LPD for `isShipOfInterest(ship)`

```

1 if any ship have ( hasTerroristPlan = NoPlan ) [
2   true = 0,
3   false = 1
4 ] else [
5   true = 1,
6   false = 0
7 ]

```

Figure B.24 presents the Meeting MFragment. Listing B.48 presents the LPD for the RV `areMeeting(ship)`. The assumption behind this LPD is that a ship of interest, with plans of exchanging illicit cargo, is more likely to meet other ships (rule 7).

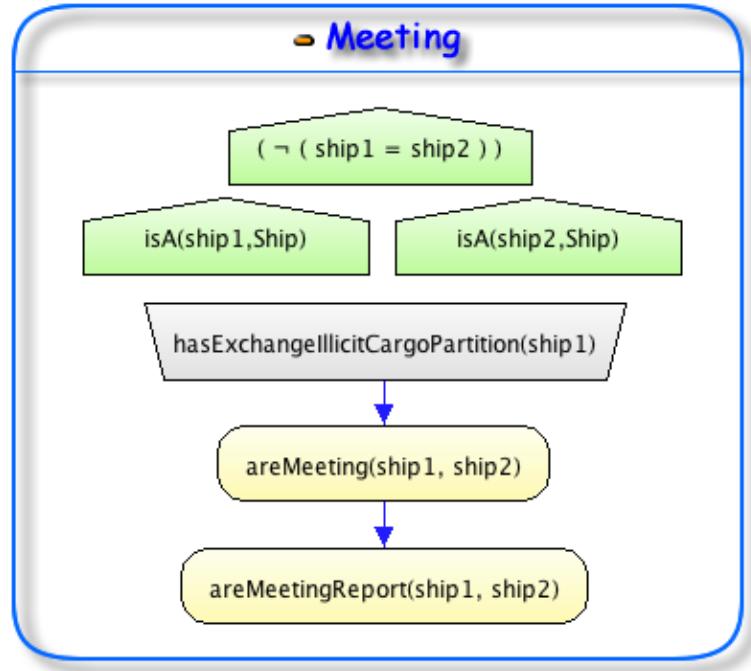


Figure B.24: Meeting MFragment.

Listing B.48: LPD for `areMeeting(ship1, ship2)`

```

1 if any ship1 have ( hasExchangeIllicitCargoPartition = NoPlan ) [
2   true = 0,
3   false = 1
4 ] else [
5   true = 1,
6   false = 0
7 ]

```

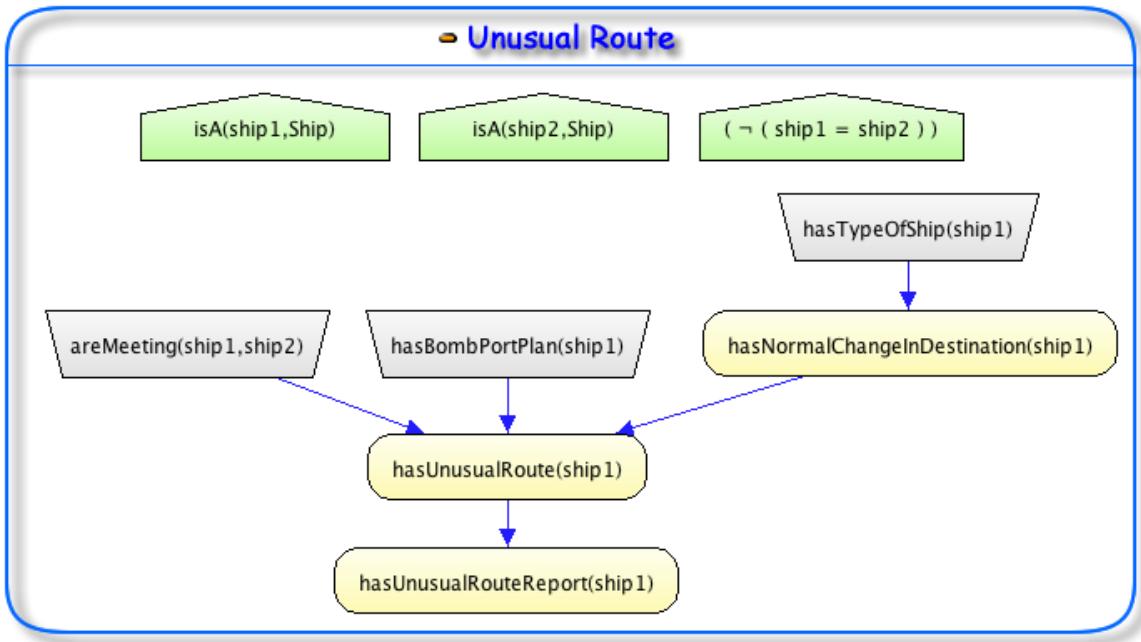


Figure B.25: Unusual Route MFragment.

Figure B.25 presents the Unusual Route MFragment. Listings B.49 and B.50 present the LPDs for the RVs `hasNormalChangeInDestination(ship1)` and `hasUnusualRoute(ship1)`, respectively. The assumptions behind these LPDs are that a fishing ship is more likely to have a normal change in its destination (*e.g.*, to sell the sh caught) than merchant ships (rule 9), that a normal change in destination will probably change the usual route of the ship (rule 10), that a ship of interest is more likely to have an unusual route, independent of its intention (rule 6), and that a ship that meets other ships to trade illicit

cargo is more likely to have an unusual route (rule 8).

Listing B.49: LPD for `hasNormalChangeInDestination(ship1)`

```

1 if any ship1 have ( hasTypeOfShip = Fishing ) [
2   true = .2,
3   false = .8
4 ] else if any ship1 have ( hasTypeOfShip = Merchant ) [
5   true = .05,
6   false = .95
7 ] else [
8   true = .1,
9   false = .9
10 ]

```

Listing B.50: LPD for `hasUnusualRoute(ship1)`

```

1 if any ship1 have ( hasBombPortPlan = true | hasNormalChangeInDestination =
2   true ) [
3   true = .9,
4   false = .1
5 ] else if any ship1.ship2 have ( areMeeting = true ) [
6   true = .9,
7   false = .1
8 ] else [
9   true = .05,
10  false = .95
]

```

Figure B.26 presents the Evasive Behavior MFragment. Listing B.51 presents the LPD for the RV `hasEvasiveBehavior(ship)`. The assumption behind this LPD is that a ship of interest, with plans of exchanging illicit cargo, is more likely to have an evasive behavior (rule 11).

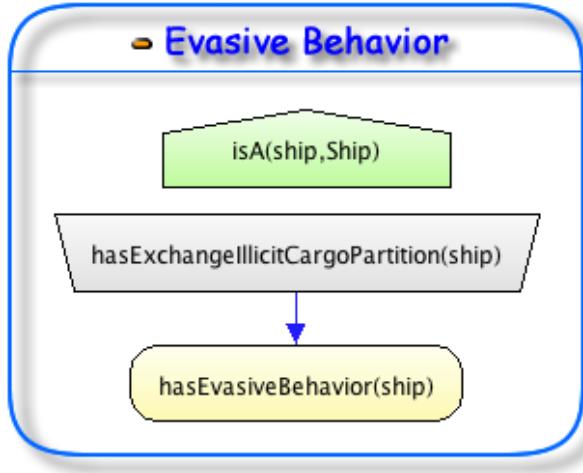


Figure B.26: Evasive Behavior MFragment.

Listing B.51: LPD for `hasEvasiveBehavior(ship)`

```

1 if any ship have ( hasExchangeIllicitCargoPartition =
2   ExchangeIllicitCargoPlanForMerchantShip ) [
3   true = 1,
4   false = 0
5 ] else [
6   true = 0,
7   false = 1
]

```

Figure B.27 presents the Aggressive Behavior MFragment. Listings B.52, B.53, and B.54 present the LPDs for the RVs `hasAggressiveBehavior(ship)`, `hasWeaponVisible(ship)`, and `isJettisoningCargo(ship)`, respectively. The assumptions behind these LPDs are that a ship of interest, independent of its intention, is more likely to have an aggressive behavior (rule 20), that a ship with aggressive behavior is more likely to have weapons visible and to jettison cargo (rule 21), and that a ship with normal behavior is not likely to have weapons visible nor to jettison cargo (rule 22).

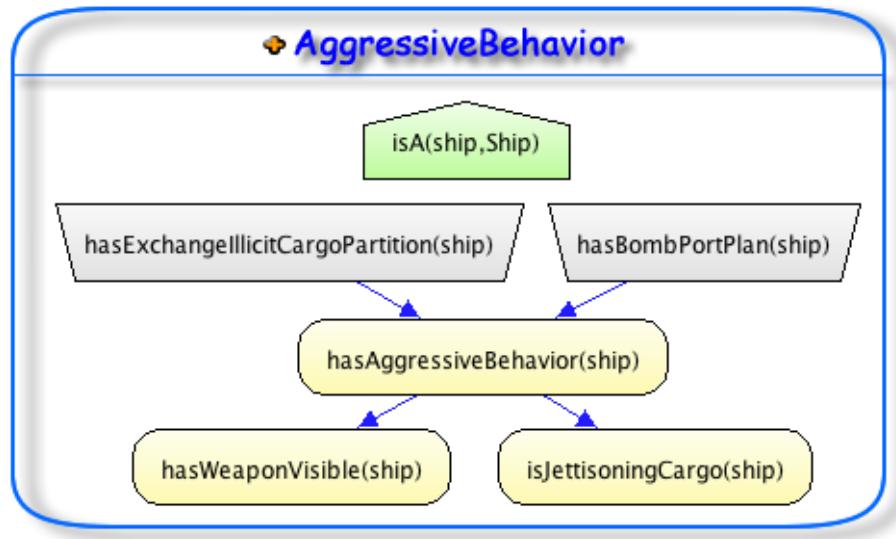


Figure B.27: Aggressive Behavior MFragment.

Listing B.52: LPD for hasAggressiveBehavior(ship)

```

1 if any ship have ( ~ hasExchangeIllicitCargoPartition = NoPlan |
2     hasBombPortPlan = true ) [
3     true = 1,
4     false = 0
5 ] else [
6     true = 0,
7     false = 1
]

```

Listing B.53: LPD for hasWeaponVisible(ship)

```

1 if any ship have ( hasAggressiveBehavior = true ) [
2     true = .7,
3     false = .3
4 ] else [
5     true = .05,
6     false = .95
7 ]

```

Listing B.54: LPD for `isJettisoningCargo(ship)`

```

1 if any ship have ( hasAggressiveBehavior = true ) [
2   true = .25,
3   false = .75
4 ] else [
5   true = .05,
6   false = .95
7 ]

```

Figure B.28 presents the Erratic Behavior MFragment. Listings B.55, B.56, and B.57 present the LPDs for the RVs `hasErraticBehavior(ship)`, `hasEquipmentFailure(ship)`, and `isCrewVisible(ship)`, respectively. The assumptions behind these LPDs are that a ship of interest, with plans of exchanging illicit cargo, is more likely to have an erratic behavior (rule 16), that a ship with normal behavior usually does not have the crew visible on the deck (rule 17), that a ship with erratic behavior usually has the crew visible on the deck (rule 18), and that if the ship has some equipment failure, it is more likely to see the crew on the deck in order to fix the problem (rule 19).

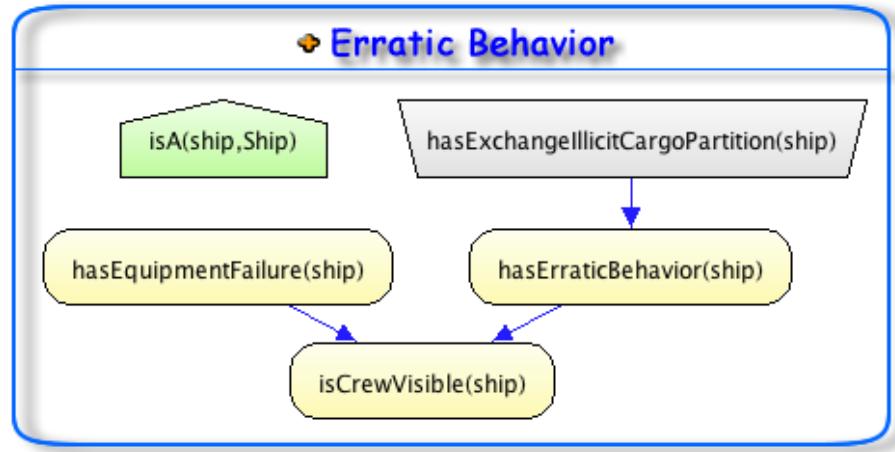


Figure B.28: Erratic Behavior MFragment.

Listing B.55: LPD for hasErraticBehavior(ship)

```
1 if any ship have ( hasExchangeIllicitCargoPartition =
2   ExchangeIllicitCargoPlanForMerchantShip ) [
3   true = 1,
4   false = 0
5 ] else [
6   true = 0,
7   false = 1
8 ]
```

Listing B.56: LPD for hasEquipmentFailure(ship)

```
1 [
2   true = .05,
3   false = .95
4 ]
```

Listing B.57: LPD for isCrewVisible(ship)

```
1 if any ship have ( hasErraticBehavior = true ) [
2   if any ship have ( hasEquipmentFailure = true ) [
3     true = .65,
4     false = .35
5   ] else [
6     true = .6,
7     false = .4
8   ]
9 ] else if any ship have ( hasErraticBehavior = false ) [
10   if any ship have ( hasEquipmentFailure = true ) [
11     true = .45,
12     false = .55
13   ] else [
14     true = .05,
15     false = .95
16   ]
17 ] else [
18   true = .05,
19   false = .95
20 ]
```

B.2.3 Third Iteration

Although I am the first author of the paper published at Fusion 2011 on PO for MDA [18], which most of this Subsection is based on, most of the research on the domain tackled in this iteration was done by Richard Haberlin, who is co-author of the paper and SME on the PROGNOS project. In fact, the first paper published by Haberlin and Costa about this domain was [55]. However, the model presented in it was only a BN, not a probabilistic ontology implemented using PR-OWL/MEBN.

In order to make reference easier, the rules defined during Analysis & Design in Chapter 5 Subsection 5.2.3 will be repeated here. The rules are:

1. Terrorist organization grouping;
 - (a) *If a crew member is a member of a terrorist organization, then it is more likely that he is a terrorist;*
 - (b) *If an organization has a terrorist member, it is more likely that it is a terrorist organization.*
2. Background influence grouping;
 - (a) For those who are terrorists, 100% of them chose to do so because of something in their past. That is, no one was born a terrorist, or just woke up one day and decided to be a terrorist. That is the easy case. For those who are not, 20% chose not to become terrorists despite having some possible factor in their background and 80% chose not to become a terrorist possibly because they have never been exposed³.
 - (b) An individual is usually negatively affected (leads him/her in becoming a terrorist) by having direct knowledge of someone either detained or killed by coalition forces during the conflict;

³This rule and explanation was given by the SME.

- (c) In the geographic area of interest, an estimated 2% of the population knows someone who was killed as a result of OEF/OIF [94];
- (d) In the geographic area of interest, approximately 2% of the population knows someone detained as a result of coalition operations [94];
- (e) Contrary to common perception, terrorists are predominantly married in keeping with the teachings of the Quran [116]. And about half of the general population in the target demographic is married.

3. Communication grouping:

- (a) It is possible that a crew member may communicate with a terrorist without being involved in terrorism due to non-terrorist affiliations or other relationships that have some normal expectation of interaction;
- (b) For each of the internet communications paths there is also the background usage rate of 28.8% in the Middle East [5]. Because the data is not broken down for the three internet transmission paths, this probability was applied equally to chat room, email, and weblog methods of communication;
- (c) Similarly, cellular telephone usage among the general population is assumed to be 31.6% based on Egyptian subscriber rates [4];
- (d) Given the availability of cellular technology and subscribing to the prioritization, a probability of 90% is assigned to terrorists communicating using cellular telephones;
- (e) The transient nature and unfettered availability of chat room communications makes it appealing to individuals who desire to remain nameless. A probability of 85% is assigned to terrorists communicating through chat rooms;
- (f) Email is the least desirable form of communication because it requires some form of subscriber account. Even in the event that fictitious information is used in creating such an account, an auditable trail may lead determined forces to

the originator. Still, it is a versatile means of communication and is assigned a probability of 65% for terrorists;

- (g) The anonymity associated with weblog interaction is very appealing to terrorists. This path is similar to chat room communications, but is less transient in content and can reach more subscribers simultaneously. For these reasons, a probability of 80% is assigned to weblog communications.

4. Relationship grouping;

- (a) Research shows that if a crew member has a relationship with terrorists, there is a 68% chance that he has a friend who is a terrorist;
- (b) Research shows that if a crew member has a relationship with terrorists, there is a 14% chance that he is related to a terrorist.

5. Cluster grouping;

- (a) It is assumed that all active terrorists fall into one of the terrorist cliques or their subsidiaries described by Sageman [116];
- (b) Contrary to popular thought, terrorists tend to not be unskilled drifters with no options other than martyrdom;
- (c) Many believe terrorist recruits to be uneducated simpletons who are easily persuaded by eloquent muftis who appeal to their sense of honor and perception of persecution. In fact, the data indicate that the typical terrorist is more educated than the average global citizen and is by far more educated than those in the Middle East, North Africa, and Southeastern Asia regions [116];
- (d) Terrorist from the clusters described by Sageman [116] are less likely to be of lower class than other people from that demographic area.

Rules inherited from the first and second iterations are in italic. Items crossed out refer to rules that were considered in the first and second iteration, but now they have been changed or removed.

This Section will only describe the MFrags and LPDs that have been changed or added in the third iteration. Please refer to Sections B.2.1 and B.2.2 for those that remain the same. Figure B.29 presents the MTheory created in the third iteration. The Terrorist Person MFragment was the only one that was changed. All others were added in this iteration.

Listing B.58 presents the LPD for the RV `isTerroristPerson(person)` from the Terrorist Person MFragment. The assumptions behind this LPD and the other ones not shown here because they are the same as in the previous iterations, are the rules in terrorist organization grouping.

Listing B.58: LPD for `isTerroristPerson(person)`

```

1 [ 
2   true = .001,
3   false = .999
4 ]
```

Listings B.59, B.60, B.61, B.62, and B.63 present the LPDs for the RVs `communicatesWithTerrorist(person)`, `usesCellular(person)`, `usesEmail(person)`, `usesWeblog(person)`, and `usesChatroom(person)`, respectively. These RVs are defined in the Person Communications MFragment. The assumptions behind these LPDs are the rules in communication grouping.

Listing B.59: LPD for `communicatesWithTerrorist(person)`

```

1 if any person have ( isTerroristPerson = true ) [
2   true = 1,
3   false = 0
4 ] else if any person have ( isTerroristPerson = false ) [
5   true = .001,
6   false = .999
7 ] else [
8   true = .002,
9   false = .998
10 ]
```

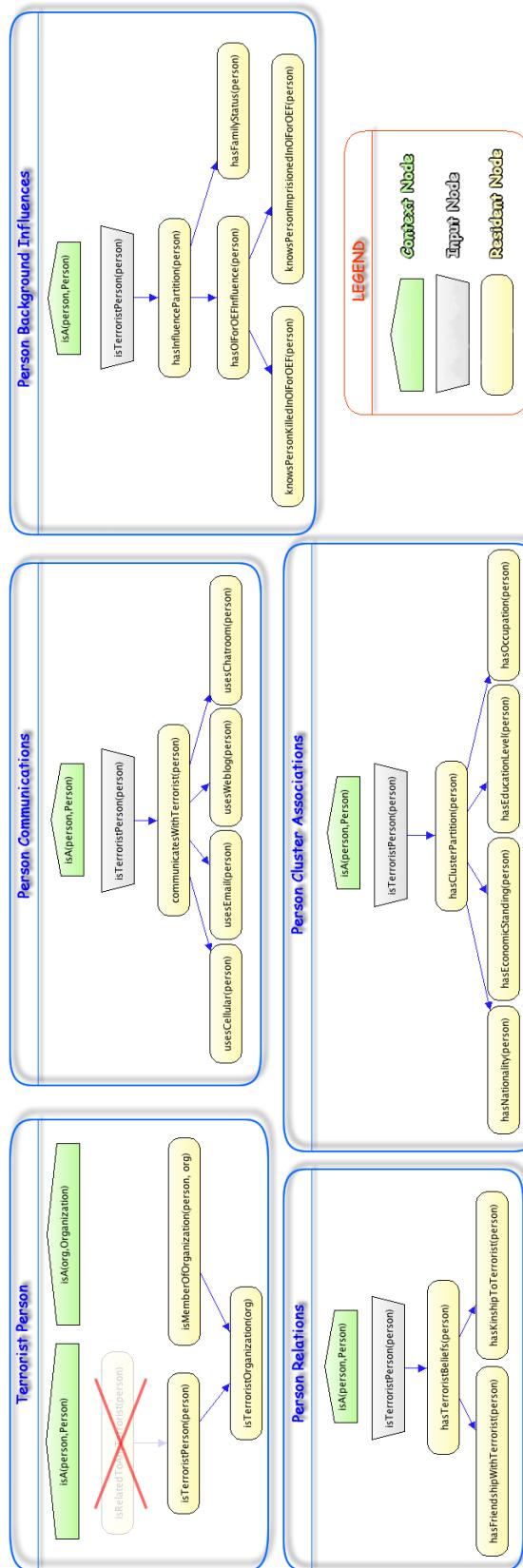


Figure B.29: MTheory created in third iteration.

Listing B.60: LPD for `usesCellular(person)`

```
1 if any person have ( communicatesWithTerrorist = true ) [
2   true = .9,
3   false = .1
4 ] else if any person have ( communicatesWithTerrorist = false ) [
5   true = .316,
6   false = .684
7 ] else [
8   true = .32,
9   false = .68
10 ]
```

Listing B.61: LPD for `usesEmail(person)`

```
1 if any person have ( communicatesWithTerrorist = true ) [
2   true = .65,
3   false = .35
4 ] else if any person have ( communicatesWithTerrorist = false ) [
5   true = .288,
6   false = .712
7 ] else [
8   true = .29,
9   false = .71
10 ]
```

Listing B.62: LPD for `usesWeblog(person)`

```
1 if any person have ( communicatesWithTerrorist = true ) [
2   true = .8,
3   false = .2
4 ] else if any person have ( communicatesWithTerrorist = false ) [
5   true = .288,
6   false = .712
7 ] else [
8   true = .29,
9   false = .71
10 ]
```

Listing B.63: LPD for `usesChatroom(person)`

```

1 if any person have ( communicatesWithTerrorist = true ) [
2   true = .85,
3   false = .15
4 ] else if any person have ( communicatesWithTerrorist = false ) [
5   true = .288,
6   false = .712
7 ] else [
8   true = .29,
9   false = .71
10 ]

```

Listings B.64, B.65, B.66, B.67, and B.68 present the LPDs for the RVs `hasInfluencePartition(person)`, `hasFamilyStatus(person)`, `hasOIForOEFInfluence(person)`, `knowsPersonKilledInOIForOEF(person)`, and `knowsPersonImpri-`
`sionedInOIForOEF(person)`,

respectively. These RVs are defined in the Person Background Influences Mfrag. The assumptions behind these LPDs are the rules in background influence grouping.

Listing B.64: LPD for `hasInfluencePartition(person)`

```

1 if any person have ( isTerroristPerson = true ) [
2   true = 1,
3   false = 0
4 ] else if any person have ( isTerroristPerson = false ) [
5   true = .20,
6   false = .80
7 ] else [
8   true = .001,
9   false = .999
10 ]

```

Listing B.65: LPD for `hasFamilyStatus(person)`

```

1 if any person have ( hasInfluencePartition = true ) [
2   Married = .73,
3   Single = .27
4 ] else if any person have ( hasInfluencePartition = false ) [
5   Married = .52,
6   Single = .48
7 ] else [
8   Married = .60,
9   Single = .40 ]

```

Listing B.66: LPD for `hasOIForOEFInfluence(person)`

```

1 if any person have ( hasInfluencePartition = true ) [
2   true = .75,
3   false = .25
4 ] else if any person have ( hasInfluencePartition = false ) [
5   true = .02,
6   false = .98
7 ] else [
8   true = .001,
9   false = .999
10 ]

```

Listing B.67: LPD for `knowsPersonKilledInOIForOEF(person)`

```

1 if any person have ( hasOIForOEFInfluence = true ) [
2   None = .98,
3   Few = .015,
4   Many = .005
5 ] else if any person have ( hasOIForOEFInfluence = false ) [
6   None = .999,
7   Few = .0008,
8   Many = .0002
9 ] else [
10   None = .999,
11   Few = .0008,
12   Many = .0002
13 ]

```

Listing B.68: LPD for `knowsPersonImprisonedInOIForOEF(person)`

```

1 if any person have ( hasOIForOEFInfluence = true ) [
2   None = .98,
3   Few = .015,
4   Many = .005
5 ] else if any person have ( hasOIForOEFInfluence = false ) [
6   None = .999,
7   Few = .0008,
8   Many = .0002
9 ] else [
10   None = .999,
11   Few = .0008,
12   Many = .0002
13 ]

```

Listings B.69, B.70, and B.71 present the LPDs for the RVs

`hasTerroristBeliefs(person)`, `hasFriendshipWithTerrorist(person)`, and `hasKinshipToTerrorist(person)`, respectively. These RVs are defined in the Person Relations Mfrag. The assumptions behind these LPDs are the rules in relationship grouping.

Listing B.69: LPD for `hasTerroristBeliefs(person)`

```

1 if any person have ( isTerroristPerson = true ) [
2   true = .75,
3   false = .25
4 ] else if any person have ( isTerroristPerson = false ) [
5   true = .001,
6   false = .999
7 ] else [
8   true = .002,
9   false = .998
10 ]

```

Listing B.70: LPD for `hasFriendshipWithTerrorist(person)`

```

1 if any person have ( hasTerroristBeliefs = true ) [
2   None = .32,
3   Few = .40,
4   Many = .28
5 ] else if any person have ( hasTerroristBeliefs = false ) [
6   None = .999,
7   Few = .0008,
8   Many = .0002
9 ] else [
10   None = .999,
11   Few = .0008,
12   Many = .0002 ]

```

Listing B.71: LPD for `hasKinshipToTerrorist(person)`

```

1 if any person have ( hasTerroristBeliefs = true ) [
2   None = .86,
3   Few = .10,
4   Many = .04
5 ] else if any person have ( hasTerroristBeliefs = false ) [
6   None = .999,
7   Few = .0008,
8   Many = .0002
9 ] else [
10   None = .999,
11   Few = .0008,
12   Many = .0002 ]

```

Listings B.72, B.73, B.74, B.75, and B.76 present the LPDs for the RVs `hasClusterPartition(person)`, `hasNationality(person)`, `hasEconomicStanding(person)`, `hasEducationLevel(person)`, and `hasOccupation(person)`, respectively. These RVs are defined in the Person Cluster Associations MFragment. The assumptions behind these LPDs are the rules in cluster grouping.

Listing B.72: LPD for `hasClusterPartition(person)`

```

1 if any person have ( isTerroristPerson = true ) [
2   CentralStaff = .18,
3   SoutheastAsia = .12,
4   MaghrebArab = .30,
5   CoreArab = .32,
6   Other = .08
7 ] else if any person have ( isTerroristPerson = false ) [
8   CentralStaff = 0,
9   SoutheastAsia = 0,
10  MaghrebArab = 0,
11  CoreArab = 0,
12  Other = 1
13 ] else [ CentralStaff = .00018,
14   SoutheastAsia = .00012,
15   MaghrebArab = .0003,
16   CoreArab = .00032,
17   Other = .99908 ]

```

Listing B.73: LPD for `hasNationality(person)`

```

1 if any person have ( hasClusterPartition = CentralStaff ) [
2   Egypt = .63,
3   SaudiArabia = .09,
4   Kuwait = .09,
5   Jordan = .06,
6   Iraq = .03,
7   Sudan = .03,
8   Libya = .03,
9   Lebanon = .04,
10  Indonesia = 0,
11  Malaysia = 0,
12  Singapore = 0,
13  Pakistan = 0,
14  Philippines = 0,
15  France = 0,
16  Algeria = 0,
17  Morocco = 0,
18  Syria = 0,
19  Tunisia = 0,
20  UAE = 0,

```

```

21 |     Yemen = 0,
22 |     Other = 0
23 | ] else if any person have ( hasClusterPartition = SoutheastAsia ) [
24 |     Egypt = 0,
25 |     SaudiArabia = 0,
26 |     Kuwait = 0,
27 |     Jordan = 0,
28 |     Iraq = 0,
29 |     Sudan = 0,
30 |     Libya = 0,
31 |     Lebannon = 0,
32 |     Indonesia = .57 ,
33 |     Malaysia = .14 ,
34 |     Singapore = .10 ,
35 |     Pakistan = 0,
36 |     Philippines = .09 ,
37 |     France = 0,
38 |     Algeria = 0,
39 |     Morocco = 0,
40 |     Syria = 0,
41 |     Tunisia = 0,
42 |     UAE = 0,
43 |     Yemen = 0,
44 |     Other = .10
45 | ] else if any person have ( hasClusterPartition = MaghrebArab ) [
46 |     Egypt = 0,
47 |     SaudiArabia = 0 ,
48 |     Kuwait = 0,
49 |     Jordan = 0,
50 |     Iraq = 0,
51 |     Sudan = 0,
52 |     Libya = 0,
53 |     Lebannon = 0,
54 |     Indonesia = 0 ,
55 |     Malaysia = 0 ,
56 |     Singapore = 0 ,
57 |     Pakistan = 0,
58 |     Philippines = 0 ,
59 |     France = .34 ,
60 |     Algeria = .28 ,
61 |     Morocco = .19 ,
62 |     Syria = 0,
63 |     Tunisia = .09 ,
64 |     UAE = 0,
65 |     Yemen = 0,
66 |     Other = .10
67 | ] else if any person have ( hasClusterPartition = CoreArab ) [
68 |     Egypt = .07 ,
69 |     SaudiArabia = .50 ,
70 |     Kuwait = .07 ,
71 |     Jordan = 0,
72 |     Iraq = 0,
73 |     Sudan = 0,
74 |     Libya = 0,
75 |     Lebannon = 0 ,
76 |     Indonesia = 0 ,

```

```

77   Malaysia = 0,
78   Singapore = 0,
79   Pakistan = .04,
80   Philippines = 0,
81   France = 0,
82   Algeria = 0,
83   Morocco = .07,
84   Syria = .04,
85   Tunisia = 0,
86   UAE = .04,
87   Yemen = .07,
88   Other = .10
89 ] else if any person have ( hasClusterPartition = Other ) [
90   Egypt = 0.0454,
91   SaudiArabia = 0.0139,
92   Kuwait = 0.0015,
93   Jordan = 0.0033,
94   Iraq = 0.0172,
95   Sudan = 0.0231,
96   Libya = 0.0035,
97   Lebannon = 0.0023,
98   Indonesia = 0.1257,
99   Malaysia = 0.015,
100  Singapore = 0.0027,
101  Pakistan = 0.0928,
102  Philippines = 0.0503,
103  France = 0.0342,
104  Algeria = 0.0191,
105  Morocco = 0.0175,
106  Syria = 0.0115,
107  Tunisia = 0.0057,
108  UAE = 0.0025,
109  Yemen = 0.0129,
110  Other = 0.50
111 ] else [
112   Egypt = 0.0454,
113   SaudiArabia = 0.0139,
114   Kuwait = 0.0015,
115   Jordan = 0.0033,
116   Iraq = 0.0172,
117   Sudan = 0.0231,
118   Libya = 0.0035,
119   Lebannon = 0.0023,
120   Indonesia = 0.1257,
121   Malaysia = 0.015,
122   Singapore = 0.0027,
123   Pakistan = 0.0928,
124   Philippines = 0.0503,
125   France = 0.0342,
126   Algeria = 0.0191,
127   Morocco = 0.0175,
128   Syria = 0.0115,
129   Tunisia = 0.0057,
130   UAE = 0.0025,
131   Yemen = 0.0129,
132   Other = 0.50 ]

```

Listing B.74: LPD for hasEconomicStanding(person)

```

1 if any person have ( hasClusterPartition = CentralStaff ) [
2   UpperClass = .35,
3   MiddleClass = .50,
4   LowerClass = .15
5 ] else if any person have ( hasClusterPartition = SoutheastAsia ) [
6   UpperClass = 0,
7   MiddleClass = .83,
8   LowerClass = .17
9 ] else if any person have ( hasClusterPartition = MaghrebArab ) [
10   UpperClass = 0,
11   MiddleClass = .52,
12   LowerClass = .48
13 ] else if any person have ( hasClusterPartition = CoreArab ) [
14   UpperClass = .29,
15   MiddleClass = .51,
16   LowerClass = .20
17 ] else if any person have ( hasClusterPartition = Other ) [
18   UpperClass = .20,
19   MiddleClass = .30,
20   LowerClass = .50
21 ] else [
22   UpperClass = .20,
23   MiddleClass = .30,
24   LowerClass = .50
25 ]

```

Listing B.75: LPD for hasEducationLevel(person)

```

1 if any person have ( hasClusterPartition = CentralStaff ) [
2   MiddleSchool = .04,
3   HighSchool = .04,
4   College = .04,
5   BA_BS = .64,
6   MA_MS = .04,
7   PhD = .20
8 ] else if any person have ( hasClusterPartition = SoutheastAsia ) [
9   MiddleSchool = 0,
10  HighSchool = .12,
11  College = .18,
12  BA_BS = .47,
13  MA_MS = .23,
14  PhD = 0
15 ] else if any person have ( hasClusterPartition = MaghrebArab ) [
16  MiddleSchool = .35,
17  HighSchool = .22,
18  College = .24,
19  BA_BS = .16,
20  MA_MS = .03,
21  PhD = 0
22 ] else if any person have ( hasClusterPartition = CoreArab ) [
23  MiddleSchool = .15,

```

```

24   HighSchool = .09 ,
25   College = .47 ,
26   BA_BS = .26 ,
27   MA_MS = .02 ,
28   PhD = .01
29 ] else if any person have ( hasClusterPartition = Other ) [
30   MiddleSchool = .44 ,
31   HighSchool = .20 ,
32   College = .15 ,
33   BA_BS = .10 ,
34   MA_MS = .08 ,
35   PhD = .03
36 ] else [
37   MiddleSchool = .44 ,
38   HighSchool = .20 ,
39   College = .15 ,
40   BA_BS = .10 ,
41   MA_MS = .08 ,
42   PhD = .03
43 ]

```

Listing B.76: LPD for `hasOccupation(person)`

```

1  if any person have ( hasClusterPartition = CentralStaff ) [
2    Professional = .63 ,
3    SemiSkilled = .33 ,
4    UnSkilled = .04
5  ] else if any person have ( hasClusterPartition = SoutheastAsia ) [
6    Professional = .78 ,
7    SemiSkilled = .17 ,
8    UnSkilled = .05
9  ] else if any person have ( hasClusterPartition = MaghrebArab ) [
10   Professional = .10 ,
11   SemiSkilled = .40 ,
12   UnSkilled = .50
13 ] else if any person have ( hasClusterPartition = CoreArab ) [
14   Professional = .45 ,
15   SemiSkilled = .33 ,
16   UnSkilled = .22
17 ] else if any person have ( hasClusterPartition = Other ) [
18   Professional = .05 ,
19   SemiSkilled = .30 ,
20   UnSkilled = .65
21 ] else [
22   Professional = .05 ,
23   SemiSkilled = .30 ,
24   UnSkilled = .65
25 ]

```

For more details on the probabilities assigned for the RVs in this Subsection, see Haberlin and Costa [55]. There they give the justification for the same kind of nodes but in a BN.

Bibliography

Bibliography

- [1] “The protege ontology editor and knowledge acquisition system,” <http://protege.stanford.edu/>. [Online]. Available: <http://protege.stanford.edu/>
- [2] “UnBBayes - the UnBBayes site,” <http://unbbayes.sourceforge.net/>. [Online]. Available: <http://unbbayes.sourceforge.net/>
- [3] “Object constraint language (OCL),” http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL, 1997. [Online]. Available: http://www.omg.org/technology/documents/modeling_spec_catalog.htm#OCL
- [4] “Wireless/Mobile statistics,” <http://www.mobileisgood.com/statistics.php>, 2010. [Online]. Available: <http://www.mobileisgood.com/statistics.php>
- [5] “World internet usage statistics news and world population stats,” <http://www.internetworkworldstats.com/stats.htm>, 2010. [Online]. Available: <http://www.internetworkworldstats.com/stats.htm>
- [6] L. Adelman, *Evaluating Decision Support and Expert Systems*, 1st ed. John Wiley & Sons, Incorporated, 1992.
- [7] D. Allemang and J. A. Hendler, *Semantic Web for the Working Ontologist*. Morgan Kaufmann, 2008.
- [8] F. Baader, I. Horrocks, and U. Sattler, “Description logics as ontology languages for the semantic web,” in *Mechanizing Mathematical Reasoning*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, pp. 228–248, 10.1007/978-3-540-32254-2_14. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-32254-2_14
- [9] R. Balduino, “Introduction to OpenUP (Open unified process),” The Eclipse Foundation, Tech. Rep., Aug. 2007. [Online]. Available: <http://www.eclipse.org/epf/general/OpenUP.pdf>
- [10] T. Berners-Lee, *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*, 1st ed. Harper Paperbacks, Nov. 2000.
- [11] R. Brachman and H. Levesque, *Knowledge Representation and Reasoning*, 1st ed. Morgan Kaufmann, Jun. 2004.
- [12] R. Braz, E. Amir, and D. Roth, “Lifted First-Order probabilistic inference,” in *Introduction to Statistical Relational Learning*. MIT Press, 2007. [Online]. Available: <http://l2r.cs.uiuc.edu/~danr/Papers/BrazAmRo07.pdf>

- [13] V. Bryl, C. Giuliano, L. Serafini, and K. Tymoshenko, “Supporting natural language processing with background knowledge: Coreference resolution case,” in *Proceedings of the 9th International Semantic Web Conference*, ser. ISWC’10, 2010, p. 80–95, ACM ID: 1940288. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1940281.1940288>
- [14] A. Calì and T. Lukasiewicz, “An approach to probabilistic data integration for the semantic web,” in *Uncertainty Reasoning for the Semantic Web I*, ser. Lecture Notes in Computer Science, P. da Costa, C. d’Amato, N. Fanizzi, K. Laskey, K. Laskey, T. Lukasiewicz, M. Nickles, and M. Pool, Eds. Springer Berlin / Heidelberg, 2008, vol. 5327, pp. 52–65, 10.1007/978-3-540-89765-1_4. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89765-1_4
- [15] A. Calì, T. Lukasiewicz, L. Predoiu, and H. Stuckenschmidt, “Tightly integrated probabilistic description logic programs for representing ontology mappings,” in *Foundations of Information and Knowledge Systems*, ser. Lecture Notes in Computer Science, S. Hartmann and G. Kern-Isberner, Eds. Springer Berlin / Heidelberg, 2008, vol. 4932, pp. 178–198, 10.1007/978-3-540-77684-0_14. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-77684-0_14
- [16] R. N. Carvalho, “Plausible reasoning in the semantic web using Multi-Entity bayesian networks - MEBN,” M.Sc., University of Brasilia, Brasilia, Brazil, Feb. 2008. [Online]. Available: <http://hdl.handle.net/123456789/159>
- [17] R. N. Carvalho, P. C. G. Costa, K. B. Laskey, and K. Chang, “PROGNOS: predictive situational awareness with probabilistic ontologies,” in *Proceedings of the 13th International Conference on Information Fusion*, Edinburgh, UK, Jul. 2010.
- [18] R. N. Carvalho, R. Haberlin, P. C. G. Costa, K. B. Laskey, and K. Chang, “Modeling a probabilistic ontology for maritime domain awareness,” in *Proceedings of the 14th International Conference on Information Fusion*, Chicago, USA, Jul. 2011.
- [19] R. N. Carvalho, M. Ladeira, L. Santos, S. Matsumoto, and P. C. G. Costa, “UnBBayes-MEBN: comments on implementing a probabilistic ontology tool,” in *Proceedings of the IADIS International Conference Applied Computing 2008*, ser. IADIS ’08, vol. Single. Algarve, Portugal: Nuno Guimares and Pedro Isaas, Apr. 2008, pp. 211–218.
- [20] R. N. Carvalho, M. Ladeira, L. L. Santos, S. Matsumoto, and P. C. G. Costa, “A GUI tool for plausible reasoning in the semantic web using MEBN,” in *Innovative Applications in Data Mining*, ser. Studies in Computational Intelligence. Nadia Nedjah, Luiza de Macedo Mourelle, Janusz Kacprzyk, 2009, vol. 169, pp. 17–45.
- [21] R. N. Carvalho, K. B. Laskey, and P. C. G. da Costa, “PR-OWL 2.0 - bridging the gap to OWL semantics,” in *Proceedings of the 6th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2010), collocated with the 9th International Semantic Web Conference (ISWC 2010)*, ser. URSW ’10, Nov. 2010, pp. 73–84. [Online]. Available: <http://ceur-ws.org/Vol-654/paper7.pdf>

- [22] R. N. Carvalho, K. B. Laskey, P. C. G. da Costa, M. Ladeira, L. L. Santos, and S. Matsumoto, “UnBBayes: modeling uncertainty for plausible reasoning in the semantic web,” in *Semantic Web*, gang wu ed. INTECH, Jan. 2010, pp. 1–28. [Online]. Available: <http://www.intechopen.com/articles/show/title/unbbayes-modeling-uncertainty-for-plausible-reasoning-in-the-semantic-web>
- [23] R. N. Carvalho, L. L. Santos, M. Ladeira, and P. C. G. Costa, “A GUI tool for plausible reasoning in the semantic web using MEBN,” in *Proceedings of the Seventh International Conference on Intelligent Systems Design and Applications*, ser. ISDA ’07. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2007, pp. 381–386.
- [24] P. P. Chen, “The Entity-Relationship model: Toward a unified view of data,” *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9–36, 1976. [Online]. Available: <http://portal.acm.org/citation.cfm?id=320440>
- [25] T. Coffman and S. Marcus, “Pattern classification in social network analysis: A case study,” in *Proceedings of the 2004 IEEE Aerospace Conference*, vol. 5, 2004, pp. 3162–3175.
- [26] P. Costa, M. Ladeira, R. N. Carvalho, K. Laskey, L. Santos, and S. Matsumoto, “A First-Order bayesian tool for probabilistic ontologies,” in *Proceedings of the 21st International Florida Artificial Intelligence Research Society Conference*, May 2008, pp. 631–636.
- [27] P. C. G. Costa, “Bayesian semantics for the semantic web,” PhD, George Mason University, Fairfax, VA, USA, Jul. 2005. [Online]. Available: <http://digilib.gmu.edu:8080/xmlui/handle/1920/455>
- [28] P. C. G. Costa, R. N. Carvalho, K. B. Laskey, and C. Y. Park, “Evaluating uncertainty representation and reasoning in HLF systems,” in *Proceedings of the 14th International Conference on Information Fusion*, Chicago, USA, Jul. 2011.
- [29] P. C. G. Costa, K. B. Laskey, and K. J. Laskey, “PR-OWL: a bayesian framework for the semantic web,” in *Proceedings of the First Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2005)*, Galway, Ireland, Nov. 2005. [Online]. Available: <http://digilib.gmu.edu:8080/xmlui/handle/1920/454>
- [30] ——, “Probabilistic ontologies for efficient resource sharing in semantic web services,” in *Proceedings of the Second Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2006)*, Athens, GA, USA, Nov. 2006. [Online]. Available: <http://digilib.gmu.edu:8080/xmlui/handle/1920/1735>
- [31] P. C. Costa, K. B. Laskey, and K. J. Laskey, “PR-OWL: a bayesian ontology language for the semantic web,” in *Uncertainty Reasoning for the Semantic Web I: ISWC International Workshops, URSW 2005-2007, Revised Selected and Invited Papers*. Springer-Verlag, 2008, pp. 88–107. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1485733>
- [32] P. C. G. Costa, K. B. Laskey, and K. Chang, “PROGNOS: applying probabilistic ontologies to distributed predictive situation assessment in naval operations,” in

- Proceedings of the Fourteenth International Command and Control Research and Technology Conference (ICCRTS 2009)*, Washington, D.C., USA, Jun. 2009. [Online]. Available: http://c4i.gmu.edu/~pcosta/pc_publications.html#2009iccrts
- [33] P. Costa, K. Chang, K. Laskey, and R. N. Carvalho, “A Multi-Disciplinary approach to high level fusion in predictive situational awareness,” in *Proceedings of the 12th International Conference on Information Fusion*, Seattle, Washington, USA, Jul. 2009, pp. 248–255.
 - [34] L. de Raedt, *Advances in Inductive Logic Programming*. IOS Press, 1996.
 - [35] R. M. de Souza, “UnBBayes plug-in for documenting probabilistic ontologies using UMP-ST,” B.S., University of Brasília, Brasilia, Brazil, Forthcoming.
 - [36] T. G. Dietterich, P. Domingos, L. Getoor, S. Muggleton, and P. Tadepalli, “Structured machine learning: The next ten years,” *Machine Learning*, vol. 73, no. 1, pp. 3–23, 2008. [Online]. Available: <http://www.springerlink.com.mutex.gmu.edu/content/v0623m861p73244g/>
 - [37] Z. Ding, Y. Peng, and R. Pan, “BayesOWL: uncertainty modeling in semantic web ontologies,” in *Soft Computing in Ontologies and Semantic Web*. Springer Berlin / Heidelberg, 2006, vol. 204, pp. 3–29, 10.1007/978-3-540-33473-6_1. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-33473-6_1
 - [38] M. J. Dombroski and K. M. Carley, “NETEST: estimating a terrorist network’s structure - graduate student best paper award, CASOS 2002 conference,” *Springer Netherlands*, vol. 8, no. 3, pp. 235–241, 2002. [Online]. Available: <http://dx.doi.org/10.1023/A:1020723730930>
 - [39] P. Domingos and D. Lowd, *Markov Logic: An Interface Layer for Artificial Intelligence*, 1st ed. Morgan and Claypool Publishers, Jun. 2009.
 - [40] P. Domingos, D. Lowd, S. Kok, H. Poon, M. Richardson, and P. Singla, “Just add weights: Markov logic for the semantic web,” in *Uncertainty Reasoning for the Semantic Web I*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, vol. 5327, pp. 1–25, 10.1007/978-3-540-89765-1_1. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89765-1_1
 - [41] L. Drumond and R. Girardi, “Extracting ontology concept hierarchies from text using markov logic,” in *Proceedings of the 2010 ACM Symposium on Applied Computing*, ser. SAC ’10, 2010, p. 1354–1358, ACM ID: 1774379.
 - [42] H. B. Enderton, *A Mathematical Introduction to Logic*, Second Edition, 2nd ed. Academic Press, Jan. 2001.
 - [43] D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider, “OIL: an ontology infrastructure for the semantic web,” *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 38–45, 2001.
 - [44] R. Fikes and T. Kehler, “The role of Frame-Based representation in reasoning,” *Communications of the ACM*, vol. 28, no. 9, pp. 904–920, 1985. [Online]. Available: <http://portal.acm.org.mutex.gmu.edu/citation.cfm?id=4284.4285>

- [45] T. E. Foundation, “Eclipse process framework project (EPF),” <http://www.eclipse.org/epf/general/description.php>, 2011. [Online]. Available: <http://www.eclipse.org/epf/general/description.php>
- [46] Y. Fukushige, “Representing probabilistic knowledge in the Semantic Web,” in *Proceedings of the W3C Workshop on Semantic Web for Life Sciences*, Oct. 2004. [Online]. Available: <http://www.w3.org/2004/09/13-Yoshio/PositionPaper.html>
- [47] L. Getoor, N. Friedman, D. Koller, and B. Taskar, “Learning probabilistic models of link structure,” *The Journal of Machine Learning Research*, vol. 3, p. 679–707, Mar. 2003, ACM ID: 944950. [Online]. Available: <http://portal.acm.org/citation.cfm?id=944919.944950>
- [48] L. Getoor, “An introduction to probabilistic graphical models for relational data,” *IEEE Data(base) Engineering Bulletin*, vol. 29, pp. 32–39, 2006.
- [49] R. Giugno and T. Lukasiewicz, “P-SHOQ (D): a probabilistic extension of (D) for probabilistic ontologies in the semantic web,” in *Logics in Artificial Intelligence*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2002, vol. 2424, pp. 86–97, 10.1007/3-540-45757-7_8. [Online]. Available: http://dx.doi.org/10.1007/3-540-45757-7_8
- [50] A. Gomez-Perez, O. Corcho, and M. Fernandez-Lopez, *Ontological Engineering: with Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web, First Edition*. Springer, Jul. 2004.
- [51] O. Gotel and C. Finkelstein, “An analysis of the requirements traceability problem,” in *Proceedings of the First International Conference on Requirements Engineering, 1994*, 1994, pp. 94–101.
- [52] B. N. Grosof, R. Volz, I. Horrocks, and S. Decker, “Description logic programs: Combining logic programs with description logic,” *SSRN eLibrary*, 2003. [Online]. Available: <http://ssrn.com/abstract=460986>
- [53] W. O. W. Group, “OWL 2 web ontology language document overview,” <http://www.w3.org/TR/2009/PR-owl2-overview-20090922/>, Sep. 2009. [Online]. Available: <http://www.w3.org/TR/2009/PR-owl2-overview-20090922/>
- [54] T. R. Gruber, “Toward principles for the design of ontologies used for knowledge sharing,” *International Journal of Human-Computer Studies - Special Issue: The Role of Formal Ontology in the Information Technology*, vol. 43, no. 5-6, pp. 907–928, 1995. [Online]. Available: <http://portal.acm.org/citation.cfm?id=219701>
- [55] R. Haberlin and P. C. G. da Costa, “A bayesian model for determining crew affiliation with terrorist organizations,” in *Proceedings of the Quantitative Methods in Defense and National Security 2010*, Fairfax, VA, USA, May 2010.
- [56] R. Haberlin, P. C. G. da Costa, and K. B. Laskey, “Hypothesis management in support of inferential reasoning,” in *Proceedings of the Fifteenth International Command and Control Research and Technology Symposium*, Santa Monica, CA, USA, Jun. 2010.

- [57] ——, “A Model-Based systems engineering approach to hypothesis management,” in *Proceedings of the Third International Conference on Model-Based Systems Engineering*, Fairfax, VA, USA, Sep. 2010.
- [58] J. Y. Halpern, “An analysis of First-Order logics of probability,” *Artificial Intelligence*, vol. 46, no. 3, pp. 311–350, Dec. 1990. [Online]. Available: <http://www.sciencedirect.com/science/article/B6TYF-47YRKR8-6T/2/d1825eac77896c1aca74b64f8930d348>
- [59] P. Hayes and A. Rector, “Defining n-ary relations on the semantic web,” <http://www.w3.org/TR/swbp-n-aryRelations/>, 2006. [Online]. Available: <http://www.w3.org/TR/swbp-n-aryRelations/>
- [60] J. Heflin, “OWL web ontology language use cases and requirements,” <http://www.w3.org/TR/2004/REC-webont-req-20040210/>, Feb. 2004, W3C Recommendation. [Online]. Available: <http://www.w3.org/TR/2004/REC-webont-req-20040210/>
- [61] J. Heinsohn, “Probabilistic description logics,” in *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI-94)*. Seattle, Washington, USA: Morgan Kaufmann, 1994, pp. 311–318. [Online]. Available: http://uai.sis.pitt.edu/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=518&proceeding_id=10
- [62] P. Hitzler, M. Krtzsch, and S. Rudolph, *Foundations of Semantic Web Technologies*, 1st ed. Chapman and Hall/CRC, Aug. 2009.
- [63] M. Holí and E. Hyvnen, “Modeling uncertainty in semantic web taxonomies,” in *Soft Computing in Ontologies and Semantic Web*, ser. Studies in Fuzziness and Soft Computing, Z. Ma, Ed. Springer Berlin / Heidelberg, 2006, vol. 204, pp. 31–46, 10.1007/978-3-540-33473-6_2. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-33473-6_2
- [64] M. Horridge and P. F. Patel-Schneider, “OWL 2 web ontology language manchester syntax,” W3C, Tech. Rep., 2009. [Online]. Available: <http://www.w3.org/2007/OWL/wiki/ManchesterSyntax>
- [65] I. Horrocks, “DAML OIL: a description logic for the semantic web,” *IEEE Data Engineering Bulletin*, vol. 25, pp. 4–9, 2002. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.1247>
- [66] I. Horrocks, P. F. Patel-Schneider, and F. V. Harmelen, “From SHIQ and RDF to OWL: the making of a web ontology language,” *Journal of Web Semantics*, vol. 1, no. 1, pp. 7–26, 2003. [Online]. Available: <http://www.citeulike.org/user/masaka/article/956280>
- [67] I. Horrocks, U. Sattler, and S. Tobies, “Reasoning with individuals for the description logic SHIQ,” in *Proceedings of the 17th International Conference on Automated Deduction*, ser. CADE-17. London, UK: Springer-Verlag, 2000, pp. 482–496. [Online]. Available: <http://portal.acm.org/citation.cfm?id=648236.753643>
- [68] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Addison-Wesley Professional, Feb. 1999.

- [69] M. Jaeger, “Relational bayesian networks,” in *Proceedings of the 13th UAI*. Morgan Kaufmann, 1997, pp. 266–273. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.435>
- [70] ——, “On the complexity of inference about probabilistic relational models,” *Artificial Intelligence*, vol. 117, no. 2, p. 297308, Mar. 2000. [Online]. Available: <http://portal.acm.org/citation.cfm?id=331623.331644>
- [71] D. Koller, A. Levy, and A. Pfeffer, “P-CLASSIC: a tractable probabilistic description logic,” *Proceedings of AAAI-97*, pp. 390–397, 1997. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.5907>
- [72] K. B. Korb and A. E. Nicholson, *Bayesian Artificial Intelligence*, 1st ed. Chapman & Hall/CRC, Sep. 2003.
- [73] V. Krebs, “Mapping networks of terrorist cells,” *Connections*, vol. 24, no. 3, pp. 43–52, 2001. [Online]. Available: <http://www.bibsonomy.org/bibtex/2c631a0818017e58cdf3a9d9785e9b698/maksim>
- [74] P. Kruchten, *The Rational Unified Process: An Introduction*, 2nd ed. Addison-Wesley Professional, Mar. 2000.
- [75] K. B. Laskey, “Sensitivity analysis for probability assessments in bayesian networks,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 25, no. 6, pp. 901 –909, Jun. 1995.
- [76] K. B. Laskey and P. C. G. Costa, “Of starships and klingons: Bayesian logic for the 23rd century,” in *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*. Arlington, Virginia, USA: AUAI Press, 2005. [Online]. Available: http://uai.sis.pitt.edu/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1225&proceeding_id=21
- [77] K. B. Laskey, P. C. G. Costa, E. J. Wright, and K. Laskey, “Probabilistic ontology for Net-Centric fusion,” in *Proceedings of the 10th International Conference on Information Fusion, 2007*, 2007, pp. 1–8.
- [78] K. B. Laskey, “MEBN: a language for First-Order bayesian knowledge bases,” *Artificial Intelligence*, vol. 172, no. 2-3, pp. 140–178, 2008. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1327646>
- [79] K. B. Laskey, P. C. G. da Costa, and T. Janssen, “Probabilistic ontologies for knowledge fusion,” in *Proceedings of the 11th International Conference on Information Fusion, 2008*, 2008, pp. 1–8.
- [80] ——, “Probabilistic ontologies for Multi-INT fusion,” George Mason University C4I Center, Tech. Rep., May 2008. [Online]. Available: <http://stinet.dtic.mil/oai/oai?&verb=getRecord&metadataPrefix=html&identifier=ADA503008>
- [81] K. B. Laskey and S. M. Mahoney, “Network engineering for agile belief network models,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 4, pp. 487–498, 2000. [Online]. Available: <http://portal.acm.org/citation.cfm?id=628073>

- [82] K. Laskey and K. B. Laskey, “Uncertainty reasoning for the world wide web: Report on the URW3-XG incubator group,” W3C, URW3-XG, 2008. [Online]. Available: http://ite.gmu.edu/~klaskey/papers/URW3_URSW08.pdf
- [83] H. J. Levesque and G. Lakemeyer, *The Logic of Knowledge Bases*, 1st ed. The MIT Press, Feb. 2001.
- [84] T. Lukasiewicz, “Probabilistic description logic programs,” *International Journal of Approximate Reasoning*, vol. 45, no. 2, pp. 288–307, 2007. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1265854>
- [85] ——, “Expressive probabilistic description logics,” *Artificial Intelligence*, vol. 172, no. 6-7, pp. 852–883, Apr. 2008. [Online]. Available: <http://www.sciencedirect.com.mutex.gmu.edu/science/article/B6TYF-4R1MF4C-1/2/36132fd965af5a169b53a197616f4721>
- [86] J. E. O. Luna, K. Revoredo, and F. G. Cozman, “Learning sentences and assessments in probabilistic description logics,” in *Proceedings of the 6th Uncertainty Reasoning for the Semantic Web (URSW 2010) on the 9th International Semantic Web Conference (ISWC 2010)*, ser. URSW ’10, Nov. 2010, pp. 85–96. [Online]. Available: <http://CEUR-WS.org/Vol-654/paper8.pdf>
- [87] ——, “Semantic query extension through probabilistic description logics,” in *Proceedings of the 6th Uncertainty Reasoning for the Semantic Web (URSW 2010) on the 9th International Semantic Web Conference (ISWC 2010)*, ser. URSW ’10, Nov. 2010, pp. 49–60. [Online]. Available: <http://CEUR-WS.org/Vol-654/paper5.pdf>
- [88] S. Matsumoto, “Framework based in plug-ins for reasoning with probabilistic ontologies,” M.Sc., University of Brasília, Brasilia, Brazil, Forthcoming.
- [89] D. L. McGuinness and F. V. Harmelen, “OWL web ontology language overview,” <http://www.w3.org/TR/owl-features/>, Feb. 2004. [Online]. Available: <http://www.w3.org/TR/owl-features/>
- [90] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov, “Blog: Probabilistic models with unknown objects,” in *Proceedings of the 19th international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 2005, pp. 1352–1359. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1642293.1642508>
- [91] B. Milch and S. Russell, “First-Order probabilistic languages: Into the unknown,” in *Inductive Logic Programming*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4455, pp. 10–24. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73847-3_3
- [92] M. Minsky, “A framework for representing knowledge,” Massachusetts Institute of Technology, Tech. Rep., 1974. [Online]. Available: <http://portal.acm.org.mutex.gmu.edu/citation.cfm?id=889222>

- [93] R. Mizoguchi, “YAMATO : Yet another more advanced top-level,” The Institute of Scientific and Industrial Research Osaka University, Tech. Rep., Dec. 2010. [Online]. Available: http://www.ei.sanken.osaka-u.ac.jp/hozo/onto_library/YAMATO.pdf
- [94] J. Moody, “Fighting a hydra: A note on the network embeddedness of the war on terror,” *Structure and Dynamics*, vol. 1, no. 2, Jan. 2005. [Online]. Available: <http://www.escholarship.org/uc/item/7x3881bs>
- [95] I. Moon and K. M. Carley, “Modeling and simulating terrorist networks in social and geospatial dimensions,” *IEEE Intelligent Systems*, vol. 22, pp. 40–49, Sep. 2007, ACM ID: 1304517. [Online]. Available: <http://dx.doi.org/10.1109/MIS.2007.91>
- [96] M. G. Morgan and M. Henrion, *Uncertainty: A Guide to Dealing with Uncertainty in Quantitative Risk and Policy Analysis*. Cambridge University Press, Jun. 1992.
- [97] B. Motik, P. F. Patel-Schneider, and B. Parsia, “OWL 2 web ontology language structural specification and Functional-Style syntax,” <http://www.w3.org/TR/owl2-syntax/>, Oct. 2009. [Online]. Available: <http://www.w3.org/TR/owl2-syntax/>
- [98] A. Mueller, “A critical study of the brazilian procurement law,” IBI - The Institute of Brazilian Business & Public Management Issues, Washington, Tech. Rep., 1998. [Online]. Available: <http://www.gwu.edu/~ibi/minerva/Fall1998/Andre.Mueller.html>
- [99] S. Muggleton, “Stochastic logic programs,” in *Advances in Inductive Logic Programming*. IOS Press, 1996, pp. 254–264.
- [100] H. Nottelmann and N. Fuhr, “Adding probabilities and rules to OWL lite subsets based on probabilistic datalog,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 14, no. 1, pp. 17–41, 2006.
- [101] N. F. Noy and D. L. McGuinness, “Ontology development 101: A guide to creating your first ontology,” Stanford Knowledge Systems Laboratory and Stanford Medical Informatics, SMI technical report SMI-2001-0880, 2001. [Online]. Available: http://www.ksl.stanford.edu/KSL_Abstracts/KSL-01-05.html
- [102] N. F. Noy, T. Tudorache, C. I. Nyulas, and M. A. Musen, “The ontology life cycle: Integrated tools for editing, publishing, peer review, and evolution of ontologies,” in *AMIA 2010 Symposium Proceedings*, Washington, DC, USA, 2010. [Online]. Available: <http://proceedings.amia.org/127gcf/1>
- [103] J. Z. Pan, G. Stoilos, G. Stamou, V. Tzouvaras, and I. Horrocks, “f-SWRL: a fuzzy extension of SWRL,” *Journal of Data Semantics VI*, vol. 4090/2006, pp. 28–46, 2006. [Online]. Available: <http://eprints.aktors.org/581/>
- [104] P. F. Patel-Schneider, P. Hayes, and I. Horrocks, “OWL web ontology language semantics and abstract syntax,” <http://www.w3.org/TR/owl-semantics/>, Feb. 2004, W3C Recommendation. [Online]. Available: <http://www.w3.org/TR/owl-semantics/>
- [105] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, 1st ed. Morgan Kaufmann, Sep. 1988.

- [106] A. Pfeffer, “IBAL: a probabilistic rational programming language,” *Proceedings of the 17TH IJCAI*, pp. 733–740, 2001. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.1299>
- [107] ——, “The design and implementation of IBAL: a generalpurpose probabilistic programming language,” *Harvard Univesity*, 2005. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.2271>
- [108] D. Poole, “Probabilistic horn abduction and bayesian networks,” *Artificial Intelligence*, vol. 64, no. 1, pp. 81–129, Nov. 1993. [Online]. Available: <http://www.sciencedirect.com/science/article/B6TYF-47YRKWW-91/2/ff00e473411cf26af878addee4645a3bd>
- [109] ——, “The independent choice logic for modelling multiple agents under uncertainty,” *Artificial Intelligence*, vol. 94, no. 1-2, pp. 7–56, Jul. 1997. [Online]. Available: <http://www.sciencedirect.com/science/article/B6TYF-3SP2BB3-2/2/dd8cd89181206276936dc31e7631afc1>
- [110] D. Poole, C. Smyth, and R. Sharma, “Semantic science: Ontologies, data and probabilistic theories,” in *Uncertainty Reasoning for the Semantic Web I*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2008, vol. 5327, pp. 26–40. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89765-1_2
- [111] L. Predoiu, “Information integration with bayesian description logic programs,” in *Proceedings of 3rd IIWeb Interdisciplinary Workshop for Information Integration on the Web in conjunction with the WWW 2006 conference*, Edinburgh, Great Britain, 2006. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.88.6485>
- [112] L. Predoiu and H. Stuckenschmidt, “A probabilistic framework for information integration and retrieval on the semantic web ABSTRACT,” in *Proceedings of the 3rd International Workshop on Database Interoperability (InterDB)*, 2007. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.2478>
- [113] ——, “Probabilistic extensions of semantic web languages - a survey,” in *The Semantic Web for Knowledge and Data Management: Technologies and Practices*. Idea Group Inc, 2008.
- [114] W. W. Royce, “Managing the development of large software systems: Concepts and techniques,” *Proceedings of IEEE WESTCON*, pp. 1–9, 1970, reprinted in Proceedings of the Ninth International Conference on Software Engineering, March 1987, pp. 328–338. [Online]. Available: <http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf>
- [115] J. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. Addison-Wesley Professional, Jan. 1999.
- [116] M. Sageman, *Understanding Terror Networks*. University of Pennsylvania Press, Apr. 2004.

- [117] T. Sato, “A glimpse of symbolic-statistical modeling by PRISM,” *Journal of Intelligent Information Systems*, vol. 31, no. 2, pp. 161–176, 2008. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1410420&dl=>
- [118] T. Sato and Y. Kameya, “New advances in Logic-Based probabilistic modeling by PRISM,” in *Probabilistic Inductive Logic Programming*. Springer-Verlag, 2008, pp. 118–155. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78652-8_5
- [119] D. A. Schum and S. Starace, *The Evidential Foundations of Probabilistic Reasoning*. Northwestern University Press, Feb. 2001.
- [120] S. Sen and A. Kruger, “Heuristics for constructing bayesian network based geospatial ontologies,” in *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4803, pp. 953–970, 10.1007/978-3-540-76848-7_63. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-76848-7_63
- [121] G. Shafer, “The construction of probability arguments,” in *Probability and Inference in the Law of Evidence*. Kluwer Academic Publishers, 1988, pp. 185–204. [Online]. Available: http://74.125.93.132/search?q=cache:UCiWL9BwiZsJ:www.glenntshafer.com/assets/downloads/articles/article26_construction88.pdf+%22The+Construction+of+Probability+Arguments%22+shafer&cd=1&hl=en&ct=clnk&gl=us&client=firefox-a
- [122] P. Singla and P. Domingos, “Lifted First-Order belief propagation,” in *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*. AAAI Press, 2008. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1620163.1620242>
- [123] M. K. Smith, C. Welty, and D. L. McGuinness, “OWL web ontology language guide,” <http://www.w3.org/TR/owl-guide/>, Feb. 2004. [Online]. Available: <http://www.w3.org/TR/owl-guide/>
- [124] I. Sommerville, *Software Engineering*, 9th ed. Addison Wesley, Mar. 2010.
- [125] D. Spiegelhalter, A. Thomas, N. Best, and W. Gilks, “BUGS 0.6 bayesian inference using gibbs sampling (Addendum to manual),” *Medical Research Council Biostatistics Unit, Institute of Public Health*, 1997. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.90.1739>
- [126] U. Straccia, “A fuzzy description logic for the semantic web,” in *Fuzzy Logic and the Semantic Web, Capturing Intelligence*. Elsevier, 2005, pp. 167–181. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.60.2720>
- [127] R. Studer, V. R. Benjamins, and D. Fensel, “Knowledge engineering: Principles and methods,” *Data & Knowledge Engineering*, vol. 25, no. 1-2, pp. 161–197, 1998. [Online]. Available: <http://www.sciencedirect.com/science/article/B6TYX-3SYXJ6S-G/2/67ea511f5600d90a74999a9fef47ac98>
- [128] J. Tao, Z. Wen, W. Hanpin, and W. Lifu, “PrDLs: a new kind of probabilistic description logics about belief,” in *New Trends in Applied Artificial*

- Intelligence*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, vol. 4570, pp. 644–654, 10.1007/978-3-540-73325-6_64. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-73325-6_64
- [129] O. Udrea, V. S. Subrahmanian, and Z. Majkic, “Probabilistic RDF,” in *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration, IRI - 2006*. Waikoloa, Hawaii, USA: IEEE Systems, Man, and Cybernetics Society, 2006, pp. 172–177.
 - [130] M. Uschold, V. R. Benjamins, B. Ch, A. Gomez-perez, N. Guarino, and R. Jasper, “A framework for understanding and classifying ontology applications,” in *Proceedings of the IJCAI99 Workshop on Ontologies*, 1999, pp. 16–21. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.6456>
 - [131] J. Vennekens, S. Verbaeten, M. Bruynooghe, and C. A, “Logic programs with annotated disjunctions,” in *Proceedigns of the International Conference on Logic Programming, 2004*, 2004, pp. 431–445. [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.69.6404>
 - [132] L. Wagenhals and A. Levis, “Course of action analysis in a cultural landscape using influence nets,” in *Proceedings of the IEEE Symposium on Computational Intelligence in Security and Defense Applications, 2007*, 2007, pp. 116–123.
 - [133] J. B. Warmer and A. G. Klappe, *The Object Constraint Language: Precise Modeling With Uml*, 1st ed. Addison-Wesley Professional, Oct. 1998.
 - [134] K. E. Wiegers, *Software Requirements*, 2nd ed. Microsoft Press, Feb. 2003.
 - [135] C. Yang and T. Ng, “Terrorism and crime related weblog social network: Link, content analysis and information visualization,” in *Proceedings of the IEEE Intelligence and Security Informatics, 2007*, 2007, pp. 55–58.
 - [136] Y. Yang and J. Calmet, “OntoBayes: an Ontology-Driven uncertainty model,” in *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-1 (CIMCA-IAWTIC'06) - Volume 01*. IEEE Computer Society, 2005, pp. 457–463. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1135162>

Curriculum Vitae

During the 3 years of his PhD, Rommel N. Carvalho, was a Graduate Research Assistant in the Department of Systems Engineering and Operations Research at George Mason University (GMU), Virginia, USA. He received his Master in Computer Science and his Bachelor of Computer Science from University of Brasilia, DF, Brazil, in 2008 and 2003, respectively. He is an Artificial Intelligence (AI) researcher with focus on uncertainty in the Semantic Web using Bayesian Inference, Data Mining, Software Engineering and Java Programming. Awarded programmer with experience in implementation of Bayesian Network systems (*e.g.* UnBBayes), Multi-Entity Bayesian Network and Probabilistic Web Ontology Language (PR-OWL), and various web-based applications. Rommel N. Carvalho has been working for the Brazilian Government at the Office of the Comptroller General (CGU) as an Information Technology (IT) expert since 2005. He has also done extensive research on fraud detection and prevention for the Brazilian Government and situation awareness for the U.S. Navy. In most of the systems he helped develop he was the project manager, which gave him the experience necessary to get the Project Management Professional (PMP) certificate. During his PhD, he has published over 15 papers, among conference and workshop papers, book chapters, journal papers, and workshop proceedings.