
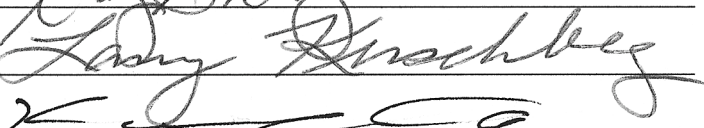

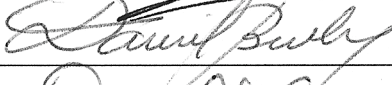
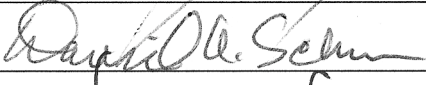





BAYESIAN SEMANTICS FOR THE SEMANTIC WEB

by

Paulo Cesar G. da Costa  
A Dissertation  
Submitted to the  
Graduate Faculty  
of  
George Mason University  
in Partial Fulfillment of  
The Requirements for the Degree  
of  
Doctor of Philosophy  
Information Technology

Committee:

	Director
	
	
	
	
	Department Chairperson
	Program Director
	Dean, School of Information Technology and Engineering

Date: JULY 12<sup>th</sup>, 2005

Summer Semester 2005  
George Mason University  
Fairfax, VA

# Bayesian Semantics for the Semantic Web

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

By

Paulo Cesar G. da Costa  
Master of Science  
George Mason University, 1999

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

Summer Semester 2005  
George Mason University  
Fairfax, VA

Copyright 2005 Paulo Cesar G. da Costa  
All Rights Reserved

## Dedication

To Claudia, whose love and support made everything possible.



## Acknowledgements

Mark Twain (1835 – 1910) once said: “Only presidents, editors and people with tapeworms have the right to use the editorial ‘we’”. However, this Dissertation can hardly be considered the result of a single’s person work, and in this section I can recognize only some of the people who helped me along the way. Thus, although I do not satisfy any of Mark Twain’s criteria, I will use the editorial ‘we’ throughout this work as a means to acknowledge the many contributors who helped me along the way in my research. In this section I recognize a few contributions that deserve special emphasis.

First among all, I would like to thank Kathryn Laskey for wearing so many hats in the last two years. When I asked her to be my advisor I already knew she was an outstanding professor and gifted researcher. However, as we worked together I began to realize how incredibly fortunate I was to have also met a mentor, co-author, and friend. She not only guided me through the many areas of knowledge I had to visit, but also showed a sensibility that few possess in pushing an advisee to achieve way beyond what he thought he could ever do. Thank you so much for your splendid support and guidance!

I was also fortunate enough to have been blessed with one of the finest committees any PhD candidate could hope to have. Following the random order in which they signed this Dissertation, I had the privilege to have Larry Kerschberg’s impressive background and always-thoughtful insights as invaluable assets I exploited many times during

my research. I could also rely on tremendously rewarding guidance from Ken Laskey, whose clear thinking and deep knowledge of the W3C, Semantic Web and Data Interchange cannot be found in any book. Even if such a book should exist some day, it will probably not match Ken's ability to make even the hardest problems look embarrassingly simple. Nonetheless, when the hardest problems came from the data mining area I was always at ease, for I could count on Daniel Barbará's extensive knowledge and experience in this field. These are dwarfed only by his talent for integrating ideas while presenting the big picture within which the real problems are best understood. The last committee member to sign the page was Dave Schum, who also signed my MS Thesis in 1999 in the same position. Thus, I was fortunate enough to have twice the opportunity to learn from such a brilliant researcher and larger than life character.

During my PhD studies I also had the chance to interact with top-notch educators who had great influence on my research. One is Tomasz Arciszewski, whose teachings in inventive engineering played a key role in my own approach to difficult challenges. I am also in debt to Carlotta Domeniconi, who took me from ground zero in data mining to a level in which I could understand how its algorithms can help to solve many problems in the Semantic Web. Finally, Dennis Buede, my advisor from the MS years, introduced me to the field of decision-making and was a major collaborator on the Wise Pilot system depicted in Appendix C. I also wish to thank Eswar Sivaraman for constructive criticism that helped me to improve my presentation skills.

I would also like to thank Tod Levitt for his contribution as a reviewer of this dissertation and some of the papers that resulted from my research. My gratitude is two-

fold, since as CEO of IET he permitted me to use their flagship product, Quiddity\*Suite, and to interact with the amazingly competent team he leads. Among that team I must personally thank Ed Wright, who made my learning path to Quiddity\*Suite much easier with his always prompt and clear advice; Mike Pool, whose expertise in OWL and excitement about the possibilities of probabilistic reasoning were both helpful and enlightening; and Masami Takikawa and Francis Fung, whose technical excellence and creativity made them outstanding co-authors.

The GMU Bayesian Group was an incredible source of enlightening discussions and a forum that helped me not only to improve my research but also to enjoy and understand the value of sharing knowledge among different research interests and areas. Among that group, special thanks go to Chris Cuppan, Aleks Lazarevich, Sepideh Mirza, Andy Powell and Mehul Revankar for reading and commenting on this work and the related papers. Ghazi Alghamdi, Stephen Cannon, Sajjad Haider, Jim Jones, Tom Shackelford, and Ning Xu also have my gratitude for the help and support and friendship they provided throughout the research period.

None of my research would have been possible without the support from the Brazilian Air Force, whose commander, General Luiz Carlos da Silva Bueno, assigned me to such a challenging endeavor. I must also recognize the support I have received from Lieutenant Generals Adenir Viana and Cleonilson Nicássio; from Colonel Milton Casimiro, and from my dear colleagues, Lt. Cols. Carlos Liberato and Tomaz Gustavo, who certainly had to face a greater workload in my absence. I am also grateful for the remarkable support I received from the Brazilian Air and Defense Attaché in

Washington, D.C., Major General Sérgio Freitas, from his Adjunct, Col. Lima de Andrade, and from Master Sargent João Luiz, Alzira Welle and Elenice Gaspar.

My family was a strong point of support in those years. I thank my dear brother Ricardo Costa for his many visits to our house, which reminded us of the wonderful relatives we have in Brazil. I also thank my sister in law, Livia Costa, for visiting us, and my brothers in law, Marclei Neves and Cleber Júnior for their steady support.

To my parents, Quintino and Vitoria Costa, and to my parents in law, Cleber and Margo Neves, I must apologize for having stolen their grandchildren for two very long years. I am sure they more than anyone have been eagerly awaiting my graduation day. I hope they forgive me for demanding such a sacrifice.

Indeed, Paulo and Laura Costa are such an awesome pair that I am sure anyone would miss them terribly. Paulo, a terrific son who keeps reminding me how I must have been when I was 14, never complained about having an absent-minded father at an age at which a boy needs a participative one. Laura, whose energy is only matched by her sweetness, constantly used the credentials of a 7-years old to rescue her dad from the books without making him angry. Thank you both for filling our lives with joy.

Finally, my deepest gratitude goes to Claudia, my wife, best friend, and lifetime partner. During our more than seventeen years together she has never ceased to impress me with her tenacity, intelligence, and wisdom. However, during those two very busy years she has not only remained the central hub of our family she has always been, but also the one who comforted, helped, and inspired all of us during the hardest times with her endless love and devotion. To her I dedicate this Dissertation.

## Table of Contents

	Page
Abstract .....	xv
Chapter 1 A Deterministic Model of a Probabilistic World .....	17
1.1 From Information to Knowledge .....	19
1.1.1 Is Semantic Information Really Important? .....	19
1.1.2 The Semantic Web and ontologies .....	23
1.2 Issues on Representing and Reasoning Using Ontologies .....	26
1.3 Why Uncertainty Matters .....	30
1.4 Research Contributions and Structure of this Dissertation .....	33
Chapter 2 Background and Related Research .....	37
2.1 Web Languages .....	37
2.2 A Brief Introduction to Probabilistic Representations .....	39
2.3 Bayesian Networks .....	43
2.3.1 Probabilistic Reasoning with Bayesian Networks .....	47
2.3.2 Case Study: The Star Trek Scenario .....	47
2.4 Probabilistic Extensions to Web Languages .....	52
2.4.1 Probabilistic extensions to Description Logic .....	52
2.4.2 Probabilistic Extensions to OWL .....	54
2.5 Probabilistic Languages with near First-Order Expressive Power .....	57
Chapter 3 Multi-Entity Bayesian Networks .....	60
3.1 A More “Realistic” Sci-fi Scenario .....	61
3.2 The Basics of MFragments .....	63
3.3 Representing Recursion in MEBN Logic .....	69
3.4 Building MEBN Models with MTheories .....	74
3.5 Making Decisions with Multi-Entity Decision Graphs. ....	81
3.6 Inference in MEBN Logic .....	83
3.7 Learning from Data .....	88
3.8 MEBN Semantics. ....	94
Chapter 4 The Path to Probabilistic Ontologies .....	101
4.1 A Polymorphic Extension to MEBN .....	107
4.1.1 The Modified MTheory Definition .....	111
4.1.2 The Star Trek MTheory Revisited .....	116
4.2 Using Quiddity*Suite for Building SSBNs .....	117
4.2.1 Concepts with Direct Translation .....	119
4.2.2 Concepts with a More Complex Translation .....	124
4.2.3 Use of Comments and Other Aspects of Quiddity*Suite .....	130

Chapter 5 PR-OWL.....	132
5.1 The Overall Implementation Strategy.....	134
5.1.1 Why MEBN as the semantic basis for PR-OWL?.....	135
5.1.2 Implementation Approach.....	139
5.2 An Upper Ontology for Probabilistic Systems.....	142
5.2.1 Creating an MFrag.....	152
5.2.2 Representing a Probability Distribution.....	160
5.3 A Proposed Operational Concept for Implementing PR-OWL.....	164
Chapter 6 Conclusion and Future Work.....	168
6.1 Summary of Contributions.....	168
6.2 A Long Road with Bright Signs Ahead.....	170
Bibliography .....	173
Appendix A Source Code for The Starship Model.....	192
Appendix B Preliminary Syntax and Semantics for PR-OWL.....	224
B.1 PR-OWL Classes .....	224
B.2 PR-OWL Properties .....	239
B.3 Naming Convention (optional).....	254
B.4 PR-OWL Upper-Ontology Code .....	256
Appendix C Potential Applications for PR-OWL Outside the Semantic Web .....	300
C.1 PR-OWL for Integration Ontologies: The DTB Project .....	300
C.2 PR-OWL for Multi-Sensor Data Fusion: The Wise Pilot System.....	305

## List of Tables

Table	Page
Table 1. Conditional Probability Table for Node MDR.....	49
Table 2. Sample Parts of the Danger To Self MFRag Probability Distribution .....	68
Table 3. MEBN Elements Directly Translated into Quiddity*Suite.....	124
Table 4. Metadata Annotation Fields .....	130
Table 5. Zone_MFRag Nodes in MEBN and PR-OWL.....	157
Table 6. Classes Used in PR-OWL .....	224
Table 7. Properties Used in PR-OWL .....	239

## List of Figures

Figure	Page
Figure 1. Simplified Text Understanding .....	20
Figure 2. Simplified Text Understanding after Data Preparation .....	21
Figure 3. Law of Total Probability.....	43
Figure 4. Sample Relationships Among Three Random Variables .....	45
Figure 5. The Naïve Star Trek Bayesian Network.....	49
Figure 6. The BN to the Four-Starship Case .....	50
Figure 7. The BN for One-Starship Case with Recursion .....	51
Figure 8. The Danger To Self MFrag.....	64
Figure 9. An Instance of the Danger To Self MFrag.....	67
Figure 10. The Zone MFrag.....	70
Figure 11. SSBN Constructed from Zone MFrag .....	71
Figure 12. The Star Trek Generative MTheory .....	76
Figure 13. Equivalent MFrag Representations of Knowledge.....	78
Figure 14. The Star Trek Decision MFrag.....	82
Figure 15. SSBN for the Star Trek MTheory with Four Starships within Range.....	86
Figure 16. Parameter Learning in MEBN.....	88
Figure 17. Structure Learning in MEBN .....	91
Figure 18. SSBNs for the Parameter Learning Example.....	93
Figure 19. Typical Web Agent's Knowledge Flow – Ignoring Uncertainty .....	103
Figure 20. Typical Web Agent's Knowledge Flow – Computing Uncertainty .....	105
Figure 21. Star Trek MTheory with the Transporter MFrag – Untyped Version .....	109
Figure 22. Built-in MFrag for Typed MEBN.....	111
Figure 23. Star Trek MTheory with the Transporter MFrag – Typed Version .....	116
Figure 24. Entity Clusters of Star Trek MTheory.....	121
Figure 25. Mapping the Sensor Report Entity Cluster to a Frame.....	123
Figure 26. Zone Entity Cluster.....	126
Figure 27. Overview of a PR-OWL MTheory Concepts.....	149
Figure 28. Elements of a PR-OWL Probabilistic Ontology .....	150
Figure 29. Header of the Starship Probabilistic Ontology.....	152
Figure 30. Initial Starship Screen with Object Properties Defined.....	153
Figure 31. Zone MFrag Represented in PR-OWL .....	155
Figure 32. ZoneMD Resident Node .....	159
Figure 33. Declarative Distributions in PR-OWL.....	161
Figure 34. A Probabilistic Assignment in a PR-OWL Table .....	162
Figure 35. Snapshot of a Graphical PR-OWL Plugin .....	165



Figure 36.	The Insider Behavior Ontology (IB) .....	302
Figure 37.	The Organization and Task Ontology (OT).....	302
Figure 38.	The Insider Threat Detection Process – Initial Setup .....	303
Figure 39.	The Insider Threat Detection Process – Data Interchange.....	304
Figure 40.	The Insider Threat Detection Process – Desired Process .....	304
Figure 41.	General Track Danger Assessment Scheme .....	307
Figure 42.	Individual Track's BN Information Exchange Scheme .....	308
Figure 43.	Wise Pilot system – general scheme.....	309
Figure 44.	Wise Pilot with 4 Tracks.....	310
Figure 45.	Wise Pilot with 5 Tracks.....	311

## List of Abbreviations

AAA – Anti-Aircraft Artillery  
 A-Box – Assertional Box (DL knowledge base assertional component)  
 $\mathcal{AL}$  – Attributive Languages (family of description logic languages)  
 $\mathcal{ALC}$  – Basic  $\mathcal{AL}$  with the concept of negation added ( $C$  means complement)  
 API – Application Programming Interface  
 ARDA – Advanced Research and Development Activity ([www.ic-arda.org](http://www.ic-arda.org))  
 BN – Bayesian Network  
 CEO – Chief Executive Officer  
 CPT – Conditional Probability Table  
 DAG – Direct Acyclic Graph  
 DAML – Darpa Agency Mark-up Language  
 DARPA – US Defense Advanced Research Project Agency  
 DL – Description Logics  
 DTB – Detection of Threat Behavior  
 FOL – First-Order Logic  
 FOPC – First Order Predicate Calculus  
 GML – Generalized Markup Language  
 GMU – George Mason University ([www.gmu.edu](http://www.gmu.edu))  
 HTML – Hypertext Markup Language  
 IBN – Insider Threat Behavioral Network  
 IET – Information Extraction and Transport, Inc. ([www.iet.com](http://www.iet.com))  
 IO – Integration Ontology  
 ISO<sup>1</sup> – International Organization for Standardization  
 MEBN – Multi-Entity Bayesian Networks  
 MFrag – MEBN Fragment  
 MTheory – MEBN Theory  
 OIL – Ontology Interface Layer  
 OOBN – Object-Oriented Bayesian Networks  
 OWL – Web Ontology Language  
 PR-OWL – Probabilistic OWL  
 RDF – Resource Description Framework  
 RDFS – RDF-Schema  
 RV – Random Variables

---

<sup>1</sup> ISO is actually a word that was derived from the Greek isos, meaning "equal".

SGML – Standard Generalized Markup Language  
SHOE – Simple HTML Ontology Extensions  
SSBN – Situation Specific Bayesian Network  
SW – Semantic Web  
T-Box – Terminological Box (DL knowledge base terminology component)  
W3C – World Wide Web Consortium  
WSMO – Web Service Modeling Ontology  
WWW – World Wide Web  
XML – Extensible Markup Language

## Abstract

### BAYESIAN SEMANTICS FOR THE SEMANTIC WEB

Paulo Cesar G. da Costa, Ph.D. Student

George Mason University, 2005

Dissertation Director: Dr. Kathryn B. Laskey

Uncertainty is ubiquitous. Any representation scheme intended to model real-world actions and processes must be able to cope with the effects of uncertain phenomena.

A major shortcoming of existing Semantic Web technologies is their inability to represent and reason about uncertainty in a sound and principled manner. This not only hinders the realization of the original vision for the Semantic Web (Berners-Lee & Fischetti, 2000), but also raises an unnecessary barrier to the development of new, powerful features for general knowledge applications.

The overall goal of our research is to establish a Bayesian framework for probabilistic ontologies, providing a basis for plausible reasoning services in the Semantic Web. As an initial effort towards this broad objective, this dissertation introduces a probabilistic extension to the Web ontology language OWL, thereby creating a crucial enabling technology for the development of probabilistic ontologies.

The extended language, PR-OWL (pronounced as “prowl”), adds new definitions to current OWL while retaining backward compatibility with its base language. Thus, OWL-built legacy ontologies will be able to interoperate with newly developed probabilistic ontologies. PR-OWL moves beyond deterministic classical logic (Frege, 1879; Peirce, 1885), having its formal semantics based on MEBN probabilistic logic (Laskey, 2005).

By providing a means of modeling uncertainty in ontologies, PR-OWL will serve as a supporting tool for many applications that can benefit from probabilistic inference within an ontology language, thus representing an important step toward the World Wide Web Consortium’s (W3C) vision for the Semantic Web.

In addition, PR-OWL will be suitable for a broad range of applications, which includes improvements to current ontology solutions (i.e. by providing proper support for modeling uncertain phenomena) and much-improved versions of probabilistic expert systems currently in use in a variety of domains (e.g. medical, intelligence, military, etc).

## Chapter 1 A Deterministic Model of a Probabilistic World

We can trace attempts by humans to represent the world surrounding them to as early as 31,000 years ago, during the so called Upper Paleolithic period, where the earliest recorded cave drawings were made (Clottes *et al.*, 1995). Moving from pictures representing objects of the real world (i.e. ideograms) to pictures representing the sounds we pronounce (i.e. phonograms), humans developed the first alphabets somewhere near the twentieth century B.C.<sup>2</sup> The efficiency of written communication received a dramatic boost with the invention of the printing press by Johannes Guttenberg in 1450.

Printing had been the dominant form for representing and communicating human knowledge until the second half of the last century, when the advent of digital computing became the driving force of what Alvin Toffler (1980) called “the Third Wave” of change in human history (the first being the agricultural revolution and the second being the industrial revolution).

At this point, inquisitive readers might ask why Toffler’s terminology was chosen over the more technically oriented and widely used term “information technology revolution”.

The answer lies in the fact that we want a broader concept for the current era of changes so we can clearly distinguish the phase “information revolution”, which we

---

<sup>2</sup> Dating established by John Darnell, in his 1990s studies of rock carvings at Wadi el-Holi made by Semitic workers within the Egyptian society. For more information on alphabets and its origins see <http://www.xasa.com/wiki/en/wikipedia/a/al/alphabet.html> (as accessed in Sept 02, 2004).

consider as an almost concluded phenomenon, from “knowledge revolution”, the subsequent phase of the “Third Wave” we are experiencing nowadays.

Until the past few years, computers had been used primarily as media for storing, exchanging, and working with information. The Internet (the network infrastructure) and the World Wide Web (the information space) have played an important role as facilitators in this process. Yet, as the availability of information resources increases, we are starting to face a significant bottleneck in our ability to use it: our own capacity to process huge amounts of data.

Indeed, our cognitive process includes one extra step between receiving data and deciding and/or acting upon it, namely the need for updating our beliefs about the subject(s) of interest given the new information available to us or, in other words, to understand what the incoming data means for our decisions and actions.

In short, data per se is useless to most of our daily tasks until we transform it into knowledge. When we reach our cognitive limit for performing this task, we are experiencing what is called “information overload”.

During the “information revolution”, human beings have largely performed the transformation from data to decision-relevant knowledge, working in a data-centric scheme that we call the “information paradigm”. The “knowledge revolution” will be seen in the future as the phase in which this tedious task was successfully assigned to computers, allowing humans to shift their focus from data-centric activities to knowledge-centric activities, thus allowing them to work under the more efficient “knowledge paradigm”.

## 1.1 From Information to Knowledge

The rapid expansion of corporate computer networks and the World Wide Web (WWW) is increasing the problem of information overload, and in this race between the availability of data and our capacity of transforming it into knowledge, humanity has developed many methods for using our ever-growing computational power to make our lives easier. Yet, in spite of the many efforts in this direction, we still have to rely heavily on the human brain for breaking the information to knowledge barrier. This led us to the question: What is missing for IT techniques to be able to help us to overcome the information paradigm and begin to work under the knowledge paradigm?

We argue that the answer lies in devising ways for the computers not only to “crunch the bytes” but also to “understand” what those bytes mean. Obviously, computers don’t really understand the meaning conveyed by the bytes they “crunch”. This is just a widely used metaphor to express the idea that making semantic information explicit and computationally accessible (i.e. better organizing the structure of data) is a powerful, more elegant way of utilizing that data. In other words, if we want to extract knowledge from data we must develop technologies that allow computers to make use of semantic, contextual information attached to the data being processed.

### 1.1.1 Is Semantic Information Really Important?

Text Classification has been one of the hottest research topics in the academic community, particularly after the end of the last decade. The obvious explanation for this is the explosion of the WWW’s use since that period, where the rapid, continuously increasing availability of data is exerting a tremendous pressure to improve the capability



of knowledge retrieval technologies. The current state-of-art paradigm for text classification of a huge corpus of text data utilizes a Vector Space Representation of documents. In this scheme, text documents are transformed into a single file called “bag of words”. Then, dimensionality reduction techniques are applied to that file, which is finally subjected to knowledge retrieval techniques aimed at pointing out possible partitions of the feature space.

One limitation of most techniques based on the Bag of Words paradigm is that they fail to consider the semantic meaning of the text. That is, if two documents share roughly the same words, they will be mapped to nearby locations in the resulting space, even if they are not related to the same subject, whereas two closely related documents that do not share the same words (e.g. documents with a high use of synonyms) would be mapped in different regions. The toy example in Figure 1 illustrates this problem.

<p><b>Tr – Computer Science</b>  Before developing products for Apple’s Cocoa environment using the Xcode suite, John Grape was a well known member of the Wine Project in the Linux community, where he cultivated many admirers for his hard-working profile.</p>	<p><b>D1 – Computer Science</b>  Yellow Dog is a distribution that works in Macintosh computers, and one of its features is to allow OS X software to run natively. This is a clear contrast with Windows applications, which need to rely on emulation to run on a different OS, even in X86 computers.</p>	<p><b>D2 – Agriculture</b>  Developing countries are not known for their wine production, as their usually equatorial environment is well suited for producing cocoa, whereas grapes or even apples are much harder to cultivate.</p>
---	--	---

Figure 1. Simplified Text Understanding

Suppose we use Tr as a corpus of training data related to the class “computer science”. Then applying the usual techniques for text classification to this corpus will result in a model that can be used for classifying other documents, which will also go through the same algorithms, such as Martin Porter’s algorithm for stemming (Porter,

1980), stop word removal (remove non-descriptive words like articles, prepositions, etc), pruning infrequent words, etc. Figure 2 illustrates the kind of output that might be produced by such a system, for both the training data Tr and the data to be classified (documents D1 and D2).

<p><b>Tr – Computer Science</b>  Before developing products for Apple's Cocoa environment using the Xcode suite, John Grape was a well known member of the Wine Project in the Linux community, where he cultivated many admirers for his hard-working profile.</p>	<p><b>D1 – Computer Science</b>  Yellow Dog is a distribution that works in Macintosh computers, and one of its features is to allow OS X software to run natively. This is a clear contrast with Windows applications, which need to rely on emulation to run on a different OS, even in X86 computers.</p>	<p><b>D2 – Agriculture</b>  Developing countries are not known for their wine production, as their usually equatorial environment is well suited for producing cocoa, whereas grapes or even apples are much harder to cultivate.</p>
---	--	---

Figure 2. Simplified Text Understanding after Data Preparation

Just by inspection we can see that D1 shares only one out of its 23 words with the vocabulary within the training data Tr, which means a commonality of just 4.3%. Therefore, even though the two texts share the same subject, our word comparison algorithm would classify D1 as not being related to the class being represented by Tr, given the fact that they have few words in common.

Yet, if we do the same comparison between the training data Tr and the agriculture-related document D2, we will see that 13 out of 16 D2's words (81.3%) are also in the training data Tr, which will cause our algorithm to incorrectly classify T3 as closely related to the class being represented by Tr.

Vector Space Representation algorithms are actually used with corpuses of training data typically containing hundreds or thousands of words, instead of our toy example's 21 words. So misclassifications like the one in our three-text example are not

very likely. Indeed, as demonstrated by Sebastiani in his recent survey on automated text categorization (Sebastiani, 2002)<sup>3</sup>, syntax-based algorithms usually achieve true positive rates between 75% and 87% in text categorization problems. Still, unlikely does not mean impossible and even low error rates can be quite undesirable, especially in domains where just a few errors may be the difference between success and failure, such as terrorist screening or Intrusion Detection systems.

In the Data Mining field, the need for considering semantic information has been recognized by many researchers. There is active research into techniques aimed to extract semantic information from the data corpus itself that are focused on external data sources such as ontologies (discussed in the next session). Examples of the first group include Latent Semantic Kernels (Cristianini *et al.*, 2001), Probabilistic LSI (Hofmann, 1999), automatic cross-language retrieval (Littman *et al.*, 1997), and some variations of Kernel Methods (Joachims, 1998). In the second group we will usually find studies advocating the use of the Wordnet (Miller *et al.*, 1990; Fellbaum, 1998) as a semantic source, such as Siolas e d'Alché-Buc (2000) and Hotho *et al.* (2002, 2003).

In short, despite the successes of syntax-only algorithms, the potential increase in discrimination power that semantic information might bring must not be ignored. In highly sensitive domains such as counter-terrorism, this increase could be the key for finding the needle in the haystack without having unacceptable false alarm rates.

The former is just one example of an application of techniques for which automated incorporation of semantics would be useful. There is a widespread

---

<sup>3</sup> See table at page 47 for a direct comparison among data mining algorithms.

understanding of the importance of semantics for many information-processing applications. The following sections review two research areas closely related to this dissertation: the Semantic Web and Ontology Engineering.

### 1.1.2 The Semantic Web and ontologies

The W3C defines the Semantic Web<sup>4</sup> as a collaborative effort between the W3C itself and a large number of researchers and industrial partners that will extend the current web to provide a common framework that allows data to be shared and reused across application, enterprise, and community boundaries.

The current WWW uses markup languages<sup>5</sup> such as HTML and XML (both being “semantic-unaware” languages) as a means to convey syntax rules and conventions to extract, transform and interchange data. In this scheme, humans are the sole party responsible for dealing with the knowledge implied from that data. However, given our restrictions in dealing with huge amount of data, it is becoming not only desirable but also necessary to make use of the increasing computational power of our current machines to perform such a task. The realization of this concept is the W3C’s vision of the Semantic Web as stated by Tim Berners-Lee (Berners-Lee & Fischetti, 2000, page 177):

“...computers and networks have as their job to enable the information space ... But doesn’t it make sense ... to put their analytical power to work making sense of the vast content ... on the web? ... This creates what

---

<sup>4</sup> From the W3C Semantic Web page, <http://www.w3c.org/2001/sw/>, as extracted in June 16, 2005.

<sup>5</sup> A markup language adds computer-understandable codes (markups) to convey metadata information within a text file. Depending on the language used, this metadata can be mostly restricted to styling and layout (e.g. HTML) or also include semantic information and other advanced features (e.g. OWL).

I call a Semantic Web – a web of data that can be processed directly or indirectly by machines ... The first step is putting data on the Web in a form that machines can naturally understand...”<sup>6</sup>.

We can infer from this definition how important representing the structure of data and metadata is going to be in this new approach for the distributed information use and sharing. Indeed, the W3C further states that the Semantic Web (SW) can only reach its full potential if it becomes a place where data can be shared and processed by automated tools as well as by people.

As an example of automated tools, we can consider the case in which software agents would have the ability to perform inference on the data stored in Web sites. To do so, such agents have to “understand” the semantics of the data, in contrast to only relying on its syntax. For instance, a software agent responsible for booking a trip to Florida must be able to infer when the word “Florida” actually means the Southern State of USA, the Portuguese word meaning “decorated with flowers”, a type of large bean, or the homonymous Uruguayan province.

According to the W3C (Heflin, 2004), ontologies are envisioned as the technology providing the cement for building the Semantic Web. Ontologies contain a common set of terms for describing and representing a domain in a way that allows automated tools to use the stored data in a more context-aware fashion, intelligent software agents to afford better knowledge management, and many other possibilities brought by a standardized, more intensive use of metadata.

---

<sup>6</sup> Emphasis added.

The term Ontology was borrowed from philosophy. Its roots can be traced back to Aristotle's metaphysical studies of the nature of being and knowing<sup>7</sup>. Nonetheless, use of the term ontology in the information systems domain is relatively new, with the first appearance occurring in 1967 (Smith, 2004, page 22).

One can find many different definitions for the concept of ontology applied to information systems, each emphasizing a specific aspect its author judged as being more important. For instance, Gruber (1993) defines an ontology as a formal specification of a conceptualization or, in other words, a declarative representation of knowledge relevant to a particular domain. Uschold and Gruninger (1996) define an ontology as a shared understanding of some domain of interest. Sowa (2000, page 492) defines an ontology as a product of a study of things that exist or may exist in some domain.

With so many possibilities for defining what an ontology is, one way of avoiding ambiguity is to focus on the objectives being sought when using it. For the purposes of the present research effort, the most important aspect of ontologies is their role as a structured form of knowledge representation. Thus, our definition of ontologies is a pragmatic one that emphasizes the purposes for which ontologies are used in the Semantic Web.

**Definition 1:** An *ontology* is an explicit, formal representation of knowledge about a domain of application. This includes:

- 1.a) Types of entities that exist in the domain;
- 1.b) Properties of those entities;

---

<sup>7</sup> The term metaphysics means beyond the study of physics

1.c) Relationships among entities;

1.d) Processes and events that happen with those entities;

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. ■

Ontologies are used for the purpose of comprehensively describing knowledge about a domain in a structured and sharable way, ideally in a format that can be read and processed by a computer. The above definition can be considered as a special type of ontology, which we could label *Semantic Web Ontology*, but for the purposes of this dissertation we will use the more general term ontology.

## 1.2 Issues on Representing and Reasoning Using Ontologies

In our definition, the explicit requirement of reasoning about a given concept makes schema-oriented technologies such as XML-Schema or RDFS fall short in terms of expressiveness. For instance, a very detailed XML-Schema may include the vocabulary and the hierarchical structure of concepts within a domain of application, but still misses OWL features such as information on disjointness and uniqueness of classes, cardinality of properties<sup>8</sup>, and others that are necessary to allow inferences to be drawn from those concepts.

Similarly, as pointed out by Shelley Powers, using RDFS may allow the development of a very rich vocabulary, but it won't be as precise or as comprehensive as

---

<sup>8</sup> Some degree of cardinality exists in XML Schema

one that incorporates ontological elements from ontology languages such as OWL (Powers, 2003, page 229).

Apart from the extra expressivity that is necessary to perform reasoning with the concepts represented in an ontology, the many similarities with database schemas makes it difficult to draw a clear distinction between ontologies and database schemas. Spyns, Meersman, and Jarrar (2002) provide an interesting discussion of how the two concepts differ. They regard data models (i.e. databases, XML schemas, etc) as specifications of the structure and integrity constraints of data sets. Thus, a database schema is developed to address the specific needs and tasks for which the data set is being used, which in turn depends heavily on the enterprise being modeled.

In contrast, ontologies are intended to be applied across a broad range of tasks within a domain, and usually contain a vocabulary (terms and labels), a definition of the concepts and their respective relationships within that domain. The main objective of an ontology is to provide a formal, agreed and shared resource, which forces it to be as generic and task-independent as possible. Although an ontology typically is developed to a focused task, it is desirable for an ontology to capture rich semantic content in a manner that could be reused across tasks.

In developing a database schema, the goal is different. Schema developers focus on organizing information in ways that optimize support for the types of queries that are expected to arise in specific applications for which the database is being designed. Achieving such goal typically requires a special application to be written on top of the database mechanism that (for a relational database) implements the principles of



relational algebra. Furthermore, a database schema is typically developed under a closed world assumption, in which the only possible instances of a given relation are those implied by the objects existing in the database. (i.e. if something is not represented there then it doesn't exist).

Ontologies, on the other hand, do not necessarily carry the assumption that not being represented entails non-existence. Not having the closed world assumption means, for example, that queries about which there is insufficient information in an ontology to be proved cannot be assumed as being false. As a consequence, we should expect situations in which incomplete information within an ontology prevents a definitive answer to a query to be rather normal. This is a clear sign that uncertainty is an intrinsic component of ontology engineering, and therefore ontology languages must include sound and principled mechanisms for dealing with uncertainty.

One commonality between ontologies and database schemas is the need to provide for interoperability among systems based on different schemas and/or ontologies. In an increasingly interconnected world, the ability to exchange data as seamlessly as possible is one of the most desired features of a knowledge representation. Integrating systems created and managed by separate organizations, evolving in different scenarios, and geared to different needs and perspectives is a task that poses many challenges, even when dealing with apparently very similar structures.

To illustrate their vision of how the Semantic Web will operate, Tim Berners-Lee, James Hendler, and Ora Lassila (2001) describe a scenario in which two siblings (Pete and Lucy in the example) use SW agents to help them schedule medical appointments for

their mother. These agents perform tasks such as Web search, scheduling consolidation, constraint matching, and trust assessment. Presently, these kinds of tasks rely heavily on human intervention. According to the Semantic Web vision, automated Web agents will perform them. For this vision to be feasible, it is clear that all Web services involved must share the same meaning for the concepts involved in these activities. That is, each sibling's SW agents should treat concepts such as "20-mile radius", "appointment time", "location", "less important", etc. the same way as they are treated by the diverse Web services they would have to interact with (e.g. the doctor's Web agent, the credit card company web services, etc.).

Unfortunately, even in tightly controlled settings (e.g. small, closed environments with controlled vocabularies), semantic inconsistencies (such as different concepts with the same name, or different names for the same concept) occur frequently. Current approaches to solve this semantic mapping problem, such as enforcing compliance with standards defined by regulatory authorities (e.g. DOD directives such as 8320.1<sup>9</sup>) or employing generic matching schemes, have consistently fallen short of what is needed to realize the SW vision.

Even though some ontology languages do offer constructs that help to import one ontology into another, they lack a principled means for grading the similarity between concepts or to make plausible inferences about the mapping between them. Providing such a means is an important step towards making the semantic mapping problem a less expensive, tedious, error-prone process. In short, the lack of a principled representation

---

<sup>9</sup> Available at <http://www.defenselink.mil/nii/bpr/bprcd/0039.htm>, as of July 6, 2005.

for uncertainty in the field of ontological engineering is a major weakness hindering the efforts towards better solutions for the semantic mapping problem. More generally, lack of support for uncertainty management is a serious impediment to make the Semantic Web a reality.

### 1.3 Why Uncertainty Matters

One of the main technical differences between the current World Wide Web and the Semantic Web is that while the former relies on syntactic-only protocols, the latter adds meta-data annotations as a means to convey shared, precisely defined terms or, in other words, semantic awareness to improve the interoperability among Web resources.

From a syntactic standpoint, Grape as a fruit is equivalent to Grape as in John Grape. Semantically aware schemes must be able to represent and appropriately process differences such as this. This is not a trivial task. For semantic interoperability to work correctly we need shared sources of precisely defined concepts, which is exactly where ontologies play a key role.

Yet, when comparing two ontologies containing the term “Grape”, deterministic reasoning algorithms will either consider it to be a fruit or an undefined object (which is not the same as a non-fruit), with no intermediate grading. This is fine when complete information is available, which is frequently the case under the closed world assumption but much less common in the open world environment, where incomplete information is the rule.

In the open world case, purely logical systems may represent phenomena such as exceptions and unknown states with generic labels such as “other”, but will lose the ability to draw strong conclusions. In probabilistic systems such phenomena would carry a probabilistic qualifier, which allows valid conclusions to be drawn and also adds more flexibility to the model. There are important issues regarding open-world probabilistic reasoning that have not yet been completely addressed (c.f. Laskey & Lehner, 1994), but probabilistic systems are a promising approach for reasoning in open world.

Despite these shortcomings of logic-based systems, the current development of the future Semantic Web (which will support automated reasoning in most of its activities) is based on classical logic. For example, OWL, a W3C recommendation (Patel-Schneider *et al.*, 2004), has no built-in support for probabilistic information and reasoning, a major shortcoming for a technology that is expected to operate in a complex, open world environment.

As we will see in the next chapters, OWL has its roots in its own web language predecessors (i.e. XML, RDF), and in traditional knowledge representation formalisms that have historically not considered uncertainty. Examples of these formalisms include Frame systems (Minsky, 1975), and Description Logics, which evolved from the so-called “Structured Inheritance Networks” (Brachman, 1977).

This historical background somewhat explains the lack of support for uncertainty in OWL, a serious limitation for a language expected to support applications in an environment where one cannot simply ignore incomplete information. As an example of a similar situation in which a knowledge based system had to evolve in order to cope with

incomplete information we can refer to the Stanford University's MYCIN<sup>10</sup> project (Shortliffe *et al.*, 1975) in the medical domain.

MYCIN evolved from DENTRAL<sup>11</sup>, which was deterministic, but according to Buchanan and Shortliffe (1984, page 209): “As we began developing the first few rules for MYCIN, it became clear that the rules we were obtaining from our collaborating experts ... the inferences described were often uncertain”.

When faced with this problem of expressing this uncertainty, the initial approach adopted by most decision-making systems' developers was the subjectivist approach of probability theory (Adams, 1976). Yet, probability theory was then considered intractable so other methods were used, such as Certainty Factors (Buchanan & Shortliffe, 1984) and Dempster-Shafer's belief functions (Dempster, 1967; Shafer, 1976). These initial approaches were superseded by the development of graphical probability models. The key innovation of graphical models is the ability to express knowledge about uncertain propositions using modular components, each involving a small number of elements that can be composed into complex models for reasoning about many interrelated propositions. This ability to express knowledge as modular, local units provides major improvements in tractability, and also makes the knowledge engineering task feasible.

The graphical model formalism has been extended to other calculi such as Dempster-Shafer's belief functions, fuzzy logic, and qualitative probability. Shenoy &

---

<sup>10</sup> MYCIN was an expert system developed in the seventies to assist medical specialists in the diagnosis of infectious blood diseases, having achieved a performance comparable with that of human experts.

<sup>11</sup> DENTRAL was also an expert system in the area of mass spectrometry. Even though its events were inherently probabilistic, this was ignored by the inference engine in favor of a simpler, binary decisions about occurrence or non occurrence of those events

Demirer (2001) provide a unified graphical formalism that covers many different uncertainty calculi. Graphical probability models have made probability tractable, thus addressing the initial concerns of many researchers. Now, many medical systems use probability (Heckerman *et al.*, 1995b; Helsen & van der Gaag, 2001; Lucas *et al.*, 2001).

The evolution from deterministic reasoning to probabilistic reasoning has enabled information systems to make use of uncertain, incomplete information. This seems to be a promising path for the Semantic Web, which will inevitably confront the same uncertainty-related concerns faced by the AI field.

#### 1.4 Research Contributions and Structure of this Dissertation

Although our research is focused in the Semantic Web, we are tackling a problem that precedes even the current WWW: the quest for more efficient data exchange. Clearly, solving that problem requires more precise semantics and flexible ways to convey information. While the WWW provided a new presentation medium and technologies such as XML presented new data exchange formats, both failed to address the semantics of data being exchanged. The SW is meant to fill this gap, and the realization of its goals will require major improvements in technologies for data exchange.

Unfortunately, for historical reasons and due to the lack of expressivity of probabilistic representations in the past, current ontology languages have no built-in support for representing or reasoning with uncertain, incomplete information. In the uncertainty-laden environment in which the SW will operate, this is a major shortcoming

preventing realization of the SW vision. Indeed, in almost any domain represented in the SW there will exist a vast body of knowledge that would be completely ignored (neither represented nor reasoned upon) due to the SW language's inability to deal with it.

As a means of addressing this problem, the long-term goal of our research is to establish a Bayesian framework for probabilistic ontologies, which will provide a basis for plausible reasoning services in the Semantic Web. Clearly, the level of acceptance and standardization required for achieving this objective requires a broader effort led by the W3C, probably resulting in a W3C Recommendation formally extending the OWL language. Thus, the present dissertation should be seen as an initial effort towards that broader objective.

In the next Chapter, we provide a brief introduction to Web languages and probabilistic representations in general. Then, we change our focus to a brief coverage on the attempts to find a common ground between the SW and probabilistic representations, which also includes a view on the trend towards more expressive forms of the latter.

Chapter Three provides the necessary background on Multi-Entity Bayesian Networks (MEBN), the probabilistic first-order logic that is the mathematical backbone of PR-OWL. As a means to provide a smooth introduction to the fairly complex concepts of MEBN logic, we needed to explore a domain of knowledge that would be both easily understood and politically neutral, while still rich enough to include scenarios that would demand a highly expressive language. Thus, we constructed a running case study based

on the Star Trek<sup>12</sup> television series. Our explanations and examples assume no previous familiarity with the particulars of the Star Trek series.

We start Chapter Four with our definition of a probabilistic ontology, a key concept in our research. Then, we cover solutions to two major issues preventing the construction of probabilistic ontologies. Because MEBN was built with flexibility in mind, it has little standardization or support for many of the advanced features of OWL. This was a major obstacle for developing a MEBN-based extension to OWL, and we addressed it by developing an extended version of MEBN logic. Our version, which we explain in the first section of the Chapter, incorporates typing, polymorphism and other features that are desirable for an ontology language. In the second and last section we addressed the lack of a probabilistic reasoner that implements all the advanced features found in MEBN logic. In that section, we explain how we used Quiddity\*Suite, a powerful probabilistic toolkit developed at Information Extraction and Transport (IET), as a MEBN logic implementation and, consequently, showed its potential to be probabilistic reasoner for Semantic Web applications. More detailed aspects are conveyed in Appendix A.

In Chapter Five, we built upon the results of Chapter Four and present our results in developing PR-OWL. There, our probabilistic extension to OWL was defined as an upper probabilistic ontology, which we documented in Appendix B. We also presented an operational concept on how we foresee the use of our framework and a proposed strategy for implementing probabilistic ontologies for the Semantic Web.

---

<sup>12</sup> Star Trek and related marks are registered trademarks of Paramount Pictures.



Finally, in Chapter Six we convey a summary of this dissertation's results and present in Appendix C some possible uses of the technology proposed here for solving problems in areas outside the Semantic Web research, such as the semantic mapping and the multi-sensor data fusion problems.

Taken together, the contributions brought by this research constitute an initial step for solving the current inability of SW languages to represent uncertainty and reason under it in a principled way. Furthermore, as we suggested in the beginning of this section, these contributions also have the potential to greatly improve the efficiency with which data is exchanged, thus implying their applicability to a broader set of problems beyond the Semantic Web.

## Chapter 2 Background and Related Research

### 2.1 Web Languages

Information in the World Wide Web is encoded via markup languages, which use tags (markups) to embed metadata into a document. The concept of markup languages<sup>13</sup> was initially implemented by IBM in 1969 with the development of the Generalized Markup Language (Goldfarb, 1996), which gained in popularity throughout the seventies. Then, the growing demand for a more powerful standard led to the development of the Standard Generalized Markup Language (SGML), which was adopted as an ISO standard in 1986 (ISO:8879). SGML was a powerful language but also a very complex one, which hindered its use in popular applications.

The breakthrough that sparked the popularization of markup languages was the creation of the Hypertext Markup Language (HTML) in 1989 by Tim Berners-Lee and Robert Caillau (Connolly *et al.*, 1997). HTML is a very simple subset of SGML that is focused on the presentation of documents. It rapidly became the standard language for the World Wide Web. Yet, as the WWW became ubiquitous, the limitations of HTML became apparent, the major one being its inability to deal with data interchange due to its limited support for metadata. Even though the W3C launched new HTML versions, these were not aimed to provide support to data exchange, since HTML was not originally

---

<sup>13</sup> In spite of both being called languages, markup languages are very different from programming languages. They are static and do not process information, but only store it in a structured way.

designed for data interchange<sup>14</sup>. Even though the WWW original intent had a focus on documents, HTML's inaptitude for data interchange became a major shortcoming at the same pace the WWW became an ideal medium for data interchange.

The answer for the HTML limitations was the development of the Extensible Markup Language (XML), which is much simpler than SGML but still capable of expressing information about the contents of a document and of supporting user-defined markups. XML became a W3C recommendation in 1998. In addition to its use for data packaging (e.g. the .plist files in Mac OS X and many configuration files in Windows XP), it has become the acknowledged standard for data interchange.

With the establishment of the Semantic Web road map by the W3C in 1998, it became clear that more expressive markup languages were needed. As a result, the first Model Syntax Specification for the Resource Description Framework (RDF) was released in 1999 as a W3C recommendation. Unlike the data-centric focus of XML, RDF is intended to represent information and to exchange knowledge. Accounts of the differences between RDF and XML are widely available on the WWW (e.g. Gil & Ratnakar, 2004).

In addition to a knowledge representation language, the Semantic Web effort also needed an ontology language to support advanced Web search, software agents, and knowledge management. The latest step towards fulfilling that requirement was the release of OWL as a W3C recommendation in 2004. OWL superseded DAML+OIL

---

<sup>14</sup> HTML has a strong focus on displaying information. Even its limited, implied semantics are largely ignored. As an example, tags h1, h2, ..., h5 are commonly employed as a formatting tool, rather than to identify header levels in a document structure.

(Horrocks, 2002), a language that merged the two ontology languages being developed in the US (DAML) and Europe (OIL)<sup>15</sup>.

According to Hendler (2004), earlier languages have been used to develop tools and ontologies for specific user communities, and therefore were not defined to be compatible with the architecture of the World Wide Web in general, and the Semantic Web in particular. In contrast, OWL uses the RDF framework to provide a more general, interoperable approach by making ontologies compatible with web standards, scalable to web needs, and with the ability to be distributed across many systems. The interested reader will find information on OWL at the W3C OWL website (Miller & Hendler, 2004). Yet, as we stated before, OWL suffers from the limitations of deterministic languages and thus lacks the advantages of probabilistic reasoning.

## 2.2 A Brief Introduction to Probabilistic Representations

Schum described probability as a subject that has “a very long past but a very short history” (Schum, 1994, page 35). An abstract notion of probability may be traced back at least to Paleolithic times, in the sense that early cultures are known to have used artifacts for gambling or forecasting the future. In contrast, he adds, the first scientific works on what we now call probability theory have a more recent history, dating back to “only” 400 years ago in the pioneer writings of mathematicians Blaise Pascal (1623-1662) and Pierre de Fermat (1608-1665). It was only in the 20<sup>th</sup> century that the major

---

<sup>15</sup> The interested reader will find further information on DAML at <http://www.daml.org/> and on OIL at <http://www.ontoknowledge.org/oil/>

formal axiom systems for probability were developed (e.g. Cox, 1946; Kolmogorov, 1960/1933).

Four hundred years of scientific research and the broad acceptance of a formal axiom system have not brought a common agreement on the philosophical foundations of probability theory. Instead, many different interpretations have arisen during this time, and none has succeeded in putting an end to the discussion about what probability really is. The interested reader will find an excellent account of the historical development of the competing theories in Hacking (1975), while valuable comparative studies can be found in the works of Fine (1973), Weatherford (1982), and Cohen (1989).

The *classical* approach regards probability as the ratio of favorable cases to total, equipossible cases (Laplace, 1996/1826; Ball, 2003/1908). The *logical* approach regards probability as a logical relation between statements of evidence and hypothesis (Carnap, 1950; Keynes, 2004/1921). The *frequentist* view regards probability as the limiting frequency of successful outcomes in a long sequence of trials (von Mises, 1981/1928). The *propensity* view (Popper, 1957, 1959; Hacking, 1965; Lewis, 1980) regards probability as a physical tendency for certain events to occur. Finally, the *subjectivist* school understands probability as the degree of belief of an ideal rational agent about hypotheses for which the truth-value is unknown (Ramsey, 1931; Savage, 1972/1954; de Finetti, 1974). Despite the differences in philosophical interpretation, the mathematics is common to all approaches.

This work is related to the task of representing uncertain, incomplete knowledge that can come from diverse agents. For this reason, we adopt the subjectivist view of

probability. We have chosen subjective probability as our representation for uncertainty because of its status as a mathematically sound representation language and formal calculus for rational degrees of belief, and because it gives different agents the freedom to have different beliefs about a given hypothesis.

Although the interpretation taken in this dissertation is subjectivist, the methodology presented here is consistent with other interpretations of probability. For example, some might prefer a frequency or a propensity interpretation for probabilities that arise from processes considered to be intrinsically random. Such individuals would naturally build probabilistic ontologies only for processes they regard as intrinsically random. Others might prefer a logical interpretation of a probabilistic domain theory. In the end, the above-mentioned discussion of what probability “really is” may be better framed as an argument over what kind of applications would render justifiable the use of a probabilistic axiom system and its underlying mathematics.

Many different axiomatic formulations have been proposed that give rise to subjectivist probability as a representation for rational degrees of belief. Examples include the axiom systems of Ramsey (1931), Kolmogorov (1960/1933), Cox (1946), Savage (1972/1954), and De Finetti (de Finetti, 1990/1954). As an illustration, the following axiom system is due to Watson & Buede (1987):

- (1) For any two uncertain events,  $A$  is more likely than  $B$ , or  $B$  is more likely than  $A$ , or they are equally likely.
- (2) If  $A_1$  and  $A_2$  are any two mutually exclusive events, and  $B_1$  and  $B_2$  are any other mutually exclusive events; and if  $A_1$  is not more likely than  $B_1$ , and  $A_2$  is not more likely than  $B_2$ ; then  $(A_1 \text{ and } A_2)$  is not more likely than  $(B_1 \text{ and } B_2)$ . Further, if either  $A_1$  is less likely than  $B_1$  or  $A_2$  is less likely than  $B_2$ , then  $(A_1 \text{ and } A_2)$  is less likely than  $(B_1 \text{ and } B_2)$ .
- (3) A possible event cannot be less likely than an impossible event.

- (4) Suppose  $A_1, A_2, \dots$  is an infinite decreasing sequence of events; that is, if  $A_i$  occurs, then  $A_{i+1}$  occurs, for any  $i$ . Suppose further that  $A_i$  is not less likely than some other event  $B$ , again for any  $i$ . Then the occurrence of all the infinite set of events  $A_i, i = 1, 2, \dots$ , is not less likely than  $B$ .
- (5) There is an experiment, with a numerical outcome, such that each possible value of that outcome, in a given range, is equally likely.

All the properties of the probabilistic system used by Bayesian Networks, Influence Diagrams, and MEBN, can be derived from those axioms. Among those, two transformations are crucial for the notion of probabilistic inference: the *Law of Total Probability* and the *Bayes Rule*.

The Law of Total Probability, also known as multiplicative law (Page, 1988, page 17), gives the marginal probability distribution of a subset of random variables from joint distribution on a superset by summing over all possible values of the random variables not contained in the subset. Figure 3 illustrates the concept.

Bayes rule provides a method of updating the probability of a random variable when information is acquired about a related random variable. The standard format of Bayes rule is:

$$P(B | A) = \frac{P(A | B)P(B)}{P(A)}$$

$P(B)$  is called prior probability of  $B$ , as it reflects our belief in event  $B$  before obtaining information on event  $A$ . Likewise,  $P(B|A)$  is the posterior probability of  $B$ , and represents our new belief on event  $B$  after applying Bayes rule with the information collected from event  $A$ .

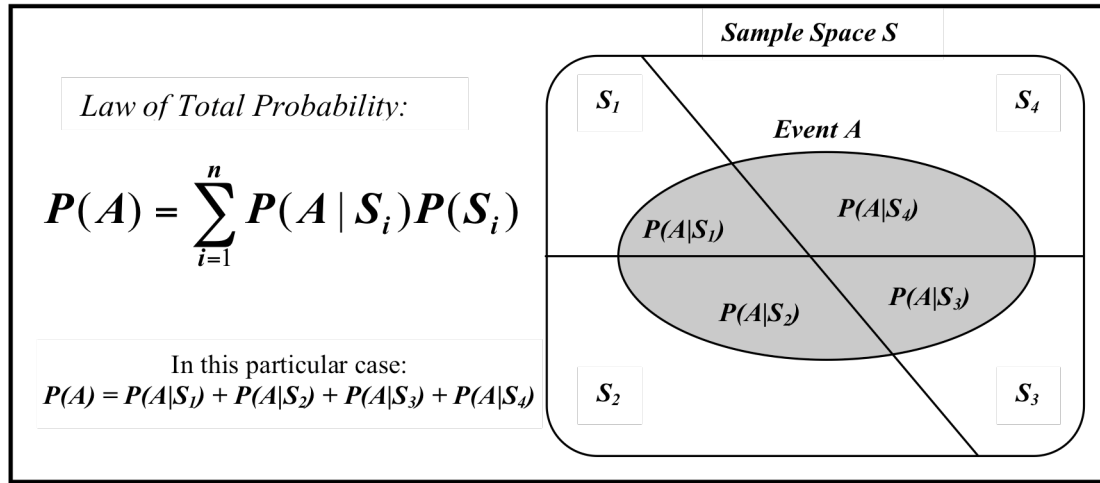


Figure 3. Law of Total Probability

Bayes rule provides the formal basis for the active and rapidly evolving field of Bayesian probability and statistics. In the Bayesian view, inference is a problem of belief dynamics. Bayes rule provides a principled methodology for belief change in the light of new information.

Good introductory material on Bayesian Statistics can be found in works of Press (1989), Lee (2004), and Gelman (2003), while a more philosophically oriented reader will be also interested in the collection of essays on foundational studies in Bayesian decision theory and statistics by Kadane *et al.* (1999). The above concepts provide the formal mathematical basis for the most widely used Bayesian Inference technique today: Bayesian Networks

### 2.3 Bayesian Networks

Bayesian networks provide a means of parsimoniously expressing joint probability distributions over many interrelated hypotheses. A Bayesian network consists of a directed acyclic graph (DAG) and a set of local distributions. Each node in the graph



represents a random variable. A random variable denotes an attribute, feature, or hypothesis about which we may be uncertain. Each random variable has a set of mutually exclusive and collectively exhaustive possible values. That is, exactly one of the possible values is or will be the actual value, and we are uncertain about which one it is. The graph represents direct qualitative dependence relationships; the local distributions represent quantitative information about the strength of those dependencies. The graph and the local distributions together represent a joint distribution over the random variables denoted by the nodes of the graph.

Bayesian networks have been successfully applied to create consistent probabilistic representations of uncertain knowledge in diverse fields such as medical diagnosis (Spiegelhalter *et al.*, 1989), image recognition (Booker & Hota, 1986), language understanding (Charniak & Goldman, 1989a, 1989b), search algorithms (Hansson & Mayer, 1989), and many others. Heckerman *et al.* (1995b) provides a detailed list of recent applications of Bayesian Networks.

One of the most important features of Bayesian networks is the fact that they provide an elegant mathematical structure for modeling complicated relationships among random variables while keeping a relatively simple visualization of these relationships. Figure 4 gives three simple examples of qualitatively different probability relationships among three random variables.

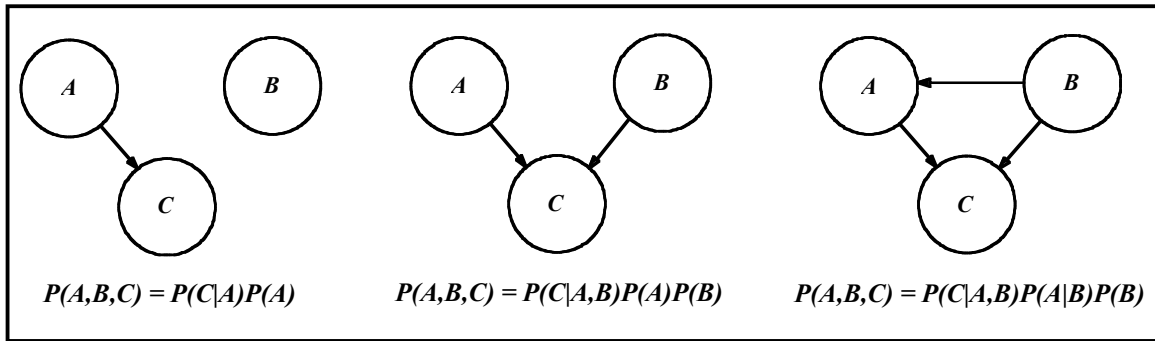


Figure 4. Sample Relationships Among Three Random Variables

As a means for realizing the communication power of this representation, one could compare two hypothetical scenarios in which a domain expert with little background in probability tries to interpret what is represented in Figure 4. Initially, suppose that she is allowed to look only to the written equations below the pictures. In this case, we believe that she will have to think at least twice before making any conclusion on the relationships among events  $A$ ,  $B$ , and  $C$ . On the other hand, if she is allowed to look only to the pictures, it seems fair to say that she will immediately perceive that in the leftmost picture, for example, event  $B$  is independent of events  $A$  and  $C$ , and event  $C$  depends on event  $A$ . Also, simply comparing the pictures would allow her to see that, in the center picture,  $A$  is now dependent on  $B$ , and that in the rightmost picture  $B$  influences both  $A$  and  $C$ . Advantages of easily interpretable graphical representation become more apparent as the number of hypothesis and the complexity of the problem increases.

One of the most powerful characteristics of Bayesian Networks is its ability to update the beliefs of each random variable via bi-directional propagation of new information through the whole structure. This was initially achieved by an algorithm

proposed by Pearl (1988) that fuses and propagates the impact of new evidence providing each node with a belief vector consistent with the axioms of probability theory.

Pearl's algorithm performs exact Bayesian updating, but only for singly connected networks. Subsequently, general Bayesian updating algorithms have been developed. One of the most commonly applied is the Junction Tree algorithm (Lauritzen & Spiegelhalter, 1988). Neapolitan (2003) provides a discussion on many Bayesian propagation algorithms. Although Cooper (1987) showed that exact belief propagation in Bayesian Networks can be NP-Hard, exact computation is practical for many problems of practical interest.

Some complex applications are too challenging for exact inference, and require approximate solutions (Dagum & Luby, 1993). Many computationally efficient inference algorithms have been developed, such as probabilistic logic sampling (Henrion, 1988), likelihood weighting (Fung & Chang, 1989; Shachter & Peot, 1990), backward sampling (Fung & del Favero, 1994), Adaptive Importance Sampling (Cheng & Druzdzel, 2000), and Approximate Posterior Importance Sampling (Druzdzel & Yuan, 2003).

Those algorithms allow the impact of evidence about one node to propagate to other nodes in multiply-connected trees, making Bayesian Networks a reliable engine for probabilistic inference. The prospective reader will find comprehensive coverage of Bayesian Networks in a large and growing literature on this subject, such as Pearl (1988), Neapolitan (1990, 2003), Oliver & Smith (1990), Charniak (1991), Jensen (1996, 2001), or Korb & Nicholson (2003).

### 2.3.1 Probabilistic Reasoning with Bayesian Networks

Bayesian Networks have received praise for being a powerful tool for performing probabilistic inference, but they do have some limitations that impede their application to complex problems.

As Bayesian networks grew in popularity, their limitations became increasingly apparent. Although a powerful tool, BNs are not expressive enough for many real-world applications. More specifically, Bayesian Networks assume a simple attribute-value representation – that is, each problem instance involves reasoning about the same fixed number of attributes, with only the evidence values changing from problem instance to problem instance.

This type of representation is inadequate for many problems of practical importance. Many domains require reasoning about varying numbers of related entities of different types, where the numbers, types and relationships among entities usually cannot be specified in advance and may have uncertainty in their own definitions. As will be demonstrated below, Bayesian networks are insufficiently expressive for such problems.

### 2.3.2 Case Study: The Star Trek Scenario

Choosing a particular real-life domain would pose the risk of getting bogged down in domain-specific detail. For this reason, we opted to construct a case study based on the popular television series *Star Trek*. Nonetheless, the examples presented here have been constructed to be accessible to anyone having some familiarity with space-based

science fiction. We begin our exposition narrating a highly simplified problem of detecting enemy starships.

In this simplified problem, the main task of a decision system is to model the problem of detecting Romulan starships (here considered as hostile by the United Federation of Planets) and assessing the level of danger they bring to our own starship, the Enterprise. All other starships are considered either friendly or neutral. Starship detection is performed by the Enterprise's suite of sensors, which can correctly detect and discriminate starships with an accuracy of 95%. However, Romulan starships may be in "cloak mode," which makes them invisible to the Enterprise's sensors. Even for the most current sensor technology, the only hint of a nearby starship in cloak mode is a slight magnetic disturbance caused by the enormous amount of energy required for cloaking. The Enterprise has a magnetic disturbance sensor, but it is very hard to distinguish background magnetic disturbance from that generated by a nearby starship in cloak mode.

This simplified situation is modeled by the BN in Figure 5<sup>16</sup>, which also considers the characteristics of the zone of space where the action takes place. Each node in our BN has a finite number of mutually exclusive, collectively exhaustive states. The node Zone Nature (ZN) is a root node, and its prior probability distribution can be read directly from Figure 5 (e.g. 80% for deep space). The probability distribution for Magnetic Disturbance Report (MDR) depends on the values of its parents ZN and Cloak Mode (CM). The strength of this influence is quantified via the conditional probability table (CPT) for

---

<sup>16</sup> Bayesian network screen shots were constructed using Netica™, <http://www.norsys.com>.

node MDR, shown in Table 1. Similarly, Operator Species (OS) depends on ZN, and the two report nodes depend on CM and the hypothesis on which they are reporting.

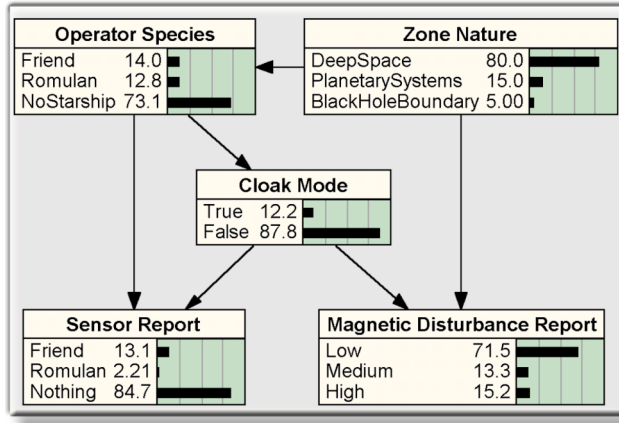


Figure 5. The Naïve Star Trek Bayesian Network

Graphical models provide a powerful modeling framework and have been applied to many real world problems involving uncertainty. Yet, the model depicted above is of little use in a “real life” starship environment. After all, hostile starships cannot be expected to approach Enterprise one at a time so as to render its simple BN model usable. If four starships were closing in on the Enterprise, the BN of Figure 5 would have to be replaced by the one shown in Figure 6.

Table 1. Conditional Probability Table for Node MDR

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

Unfortunately, building a BN for each possible number of nearby starships is not only a daunting task but also a pointless one, since there is no way of knowing in advance

how many starships the Enterprise is going to encounter and thus which BN to use at any given time. In short, BNs lack the expressive power to represent entity types (e.g., starships) that can be instantiated as many times as required for the situation at hand.

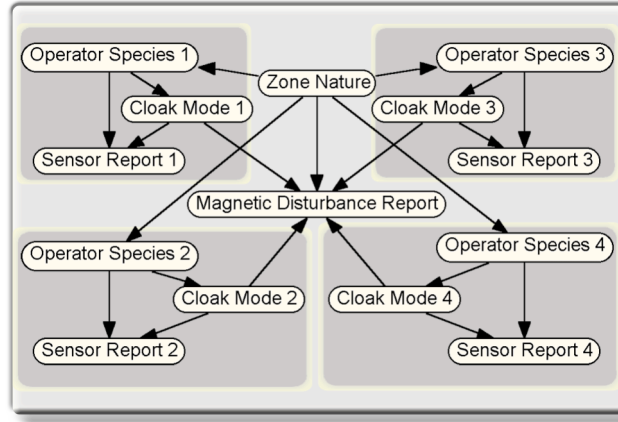


Figure 6. The BN to the Four-Starship Case

In spite of its naiveté, we will briefly hold on to the premise that only one starship can be approaching the Enterprise at a time, so that the model of Figure 5 is valid. Furthermore, we will assume that the Enterprise is traveling in deep space, and its sensor reports imply that there is no trace of any nearby starship (i.e. the state of node SR state is *Nothing*). Further, there's a newly arrived report indicating a strong magnetic disturbance (i.e. the state of node MDR is *High*). Table 1 shows that the likelihood ratio for a high MDR is  $7/5 = 1.4$  in favor of a starship in cloak mode. Although this favors a cloaked starship in the vicinity, the evidence is not overwhelming.

Repetition is a powerful way to boost the discriminatory power of weak signals. As an example from airport terminal radars, a single pulse reflected from an aircraft usually arrives back to the radar receiver very weakened, making it hard to set apart from

background noise. However, a steady sequence of reflected radar pulses is easily distinguishable from background noise.

Following the same logic, it is reasonable to assume that an abnormal background disturbance will show random fluctuation, whereas a disturbance caused by a starship in cloak mode would show a characteristic temporal pattern. Thus, when there is a cloaked starship nearby, the MDR state at any time depends on its previous state. A BN similar to the one in Figure 7 could capitalize on this for pattern recognition purposes.

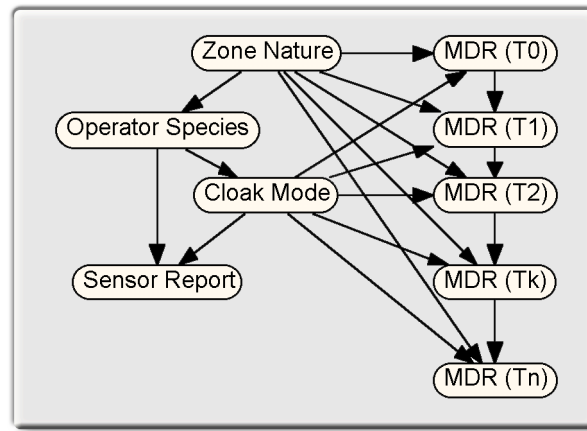


Figure 7. The BN for One-Starship Case with Recursion

Dynamic Bayesian Networks (DBNs) allow nodes to be repeated over time (Murphy, 1998). The model of Figure 7 has both static and dynamic nodes, and thus is a *partially* dynamic Bayesian network (PDBN), also known as a temporal Bayesian network (Takikawa *et al.*, 2002). While DBNs and PDBNs are useful for temporal recursion, a more general recursion capability is needed, as well as a parsimonious syntax for expressing recursive relationships.

This section has provided just a glimpse of the issues that confront an engineer attempting to apply Bayesian networks to realistically complex problems. We did not



provide a comprehensive analysis of the limitations of Bayesian networks for solving complex problems, since this brief overview is enough for making the point that even relatively simple situations might require more expressiveness than BNs can provide.

A much more powerful representational formalism is offered by first-order logic (FOL), which has the ability to represent entities of different types interacting with each other in varied ways. Sowa states that first-order logic “has enough expressive power to define all of mathematics, every digital computer that has ever been built, and the semantics of every version of logic, including itself” (Sowa, 2000, page 41). For this reason, FOL has become the *de facto* standard for logical systems from both a theoretical and practical standpoint.

However, systems based on classical first-order logic lack a theoretically principled, widely accepted, logically coherent methodology for reasoning under uncertainty. As a result, a number of languages have appeared that extend the expressiveness of standard BNs in various ways. Two different streams of research on combining logic with probability are covered in the following sections.

## 2.4 Probabilistic Extensions to Web Languages

### 2.4.1 Probabilistic extensions to Description Logic

Most of the probabilistic extensions aimed at the ontology engineering domain are based on Description Logic (DL), which Baader and Nutt (2001, page 47) define as a family of knowledge representation formalisms that represent the knowledge of an application domain (the “world”) by first defining the relevant concepts of the domain

(its terminology), and then using these concepts to specify properties of objects and individuals occurring in the domain (the world description).

Description Logic divides a knowledge base into two components: a terminological box, or T-Box, and the assertional box, or A-Box. The first introduces the terminology (i.e. the vocabulary) of an application domain, while the latter contains assertions about instances of the concepts defined in the T-Box. Description Logic is a subset of FOL that provides a very good combination of decidability and expressiveness, and is the basis of OWL-DL.

One of its extensions is Probabilistic Description Logic (Heinsohn, 1994; Jaeger, 1994), which extends the description logic  $\mathcal{ALC}$ , a member of the  $\mathcal{AL}$ -languages (Schmidt-Schauß & Smolka, 1991) that is obtained by including the full existential quantification and the union constructors to the basic  $\mathcal{AL}$  (attributive language).

Another description logic language with a probabilistic extension is SHOQ(D) (Horrocks & Sattler, 2001). SHOQ(D) is the basis of DAML+OIL (Horrocks, 2002), the language that came from merging two ontology languages being developed in the US (DAML) and Europe (OIL) and has been superseded by OWL. Its probabilistic extension is called P-SHOQ (Giugno & Lukasiewicz, 2002), and is able to represent probabilistic information about concept and role instances (i.e. A-Box).

P-Classic (Koller *et al.*, 1997), another example of a probabilistic extension to DL, uses Bayesian inference mechanisms for extending the description logic CLASSIC.

In short, each probabilistic component is associated with a set  $P$  of p-classes, and each p-class  $C$  in set  $P$  is represented using a Bayesian network.

A common characteristic of the above approaches is that they extend description logic, which is a decidable subset of first-order logic (FOL). Description logics are highly effective and efficient for the classification and assumption problems they were designed to address. However, their ability to represent and reason about other commonly occurring kinds of knowledge is limited. One restrictive aspect of DL languages is their limited ability to represent constraints on the instances that can participate in a relationship. As an example, suppose we want to express that for a starship to be a threat to another starship in a specific type of situation it is mandatory that the two individuals of class starship involved in the situation are not the same. Making sure the two starships are different in a specific situation is only possible in DL if we actually create/specify the tangible individuals involved in that situation. Indeed, stating that two “fillers” (i.e. the actual individuals of class Starship that will “fill the spaces” of concept starship in our statement) are not equal without specifying their respective values would require constructs such as *negation* and *equality role-value-maps*, which cannot be expressed in description logic. While equality role-value-maps provides additional useful means to specify structural properties of concepts, their inclusion makes the logic undecidable (Calvanese & De Giacomo, page 223).

#### 2.4.2 Probabilistic Extensions to OWL

The ontology language OWL is a W3C recommendation and has been receiving a great deal of attention, as the intended basis for the Semantic Web. Interestingly enough,

there are relatively few research efforts aimed at extending OWL to represent uncertainty.

Among those is the research being done by Zhongli Ding and Yung Peng (2004) at the University of Maryland. Their main objective is to translate a Bayesian Network model to an OWL ontology. The approach involves augmenting OWL semantics to allow probabilistic information to be represented via additional markups. The result would be a probabilistic annotated ontology that could then be translated to a Bayesian network. Such a translation would be based on a set of translation rules that would rely on the probabilistic information attached to individual concepts and properties within the annotated ontology. The authors note that after successfully achieving the translation, the resulting Bayesian network will be associated with a joint probability distribution over the application domain. The authors acknowledge that a full translation of an ontology to a standard BN is impossible given the limitations of the latter in terms of expressivity. This corroborates the comments made earlier in Section 2.3 on the limited expressiveness of Bayesian networks being a major shortcoming for the technology to be used in more complex problems.

Indeed, we have already seen that ontologies provide an explicit formal specification for how to represent the objects, concepts and other entities that are assumed to exist in some area of interest and the relationships among them, which implies the need for a highly expressive language to capture all the relevant information of a given domain. OWL, and other ontology languages as well, rely in variations of

FOL, so only a probabilistic FOL would be able to capture all the information included in an OWL ontology.

Also focusing on Bayesian extensions geared towards the Semantic Web is the work of Gu *et al.* (2004), which has a very similar approach to Ding's. Another effort in this direction is the set of RDF extensions being developed by Yoshio Fukushige (2004). In both cases, the representational limitations of Bayesian Networks limit the ability to express more complex probabilistic models, constraining their solutions to very specific classes of problems.

It is primarily in those aspects that this work differs from the above approaches. Even though we share the idea of extending Web languages to accept probabilistic information, the main objective of this research is to show a feasible solution to the more general problem of the lack of probabilistic support for applications currently being developed for the Semantic Web. At this point, it should be clear that such an objective would only be possible with the use of a powerful probabilistic language that does not have the representational limitations of Bayesian Networks.

One such approach is the work of Mike Pool at IET in developing an OWL-based implementation of Quiddity\*Suite (Pool & Aikin, 2004), which is primarily being used in the IET's KEEPER project (Pool, 2004). As much as our own work, Mike's extensions provide a very expressive method for representing uncertainty in OWL ontologies, while our approaches diverge in two aspects. First, we are focused on the more general problem of enabling probabilistic ontologies for the SW, whereas Mike's work is mostly geared towards the use of Quiddity\*Suite to represent OWL ontologies in projects such as

KEEPER. The second major difference is that while we use MEBN logic as the underlying semantics of our work, Mike relies on Quiddity\*Suite's syntax and semantics to provide the logical framework for his extensions. Given our common use of Quiddity\*Suite as a reasoner and shared interests in developing coherent forms of representing uncertainty in ontologies we have not only been mutually aware of our progresses but also had a great level of collaboration during the research period of this Dissertation.

## 2.5 Probabilistic Languages with near First-Order Expressive Power

In recent years, a number of languages have appeared that extend the expressiveness of standard directed and undirected graphical models in various ways. These languages include Hidden Markov models (Baum & Petrie, 1966; Rabiner, 1989; Elliott *et al.*, 1995) which have been largely used in pattern recognition applications. HMMs can be viewed as a special case of dynamic Bayesian networks, or DBNs (Murphy, 1998). A HMM is a DBN having hidden states with no internal structure that *d*-separate observations at different time steps. Partially dynamic Bayesian networks, also called temporal Bayesian networks (Takikawa *et al.*, 2002) extend DBNs to include static variables. These formalisms augment standard Bayesian networks with a capability for temporal recursion.

BUGS (Buntine, 1994a; Gilks *et al.*, 1994; Spiegelhalter *et al.*, 1996) is a software package that implements the Plates language. Plates represent repeated fragments of directed or undirected graphical models. Visually, a plate is represented as a rectangle

enclosing a set of repeated nodes. Strengths of plates are the ability to handle continuous distributions without resorting to discretization, and support for parameter learning in a wide variety of parameterized statistical models. The main weakness is the lack of a direct, explicit way to represent uncertainty about model structure.

Probabilistic efforts towards more expressive languages also involve undirected graph models, which are usually applied to problems where there is not a natural direction for probabilistic influences, such as image processing and terrain reasoning. One example is pattern theory (Grenander, 1995), which focuses on creating mathematical knowledge representations of complex systems, analyzing the mathematical properties or the resulting regular structures, and applying them to practically occurring patterns found in the real world. Patterns are expressed through their typical behavior as well as through their variability around their typical form, and algorithms are derived for the understanding, recognition, and restoration of the observed patterns. The theory employs undirected graphs as a representational tool.

Object-Oriented Bayesian Networks (Koller & Pfeffer, 1997; Bangsø & Wuillemin, 2000; Langseth & Nielsen, 2003) represent entities as instances of object classes with class-specific attributes and probability distributions. Probabilistic Relational Models (PRM) (Pfeffer *et al.*, 1999; Getoor *et al.*, 2000; Getoor *et al.*, 2001; Pfeffer, 2001) integrate the relational data model (Codd, 1970) and Bayesian networks. PRMs extend standard Bayesian Networks to handle multiple entity types and relationships among them, providing a consistent representation for probabilities over a relational database. PRMs cannot express arbitrary quantified first-order sentences and do not

support recursion. Although PRMs augmented with DBNs can support limited forms of recursion, they still do not support general recursive definitions. Jaeger (1997) extends relational probabilistic models to allow recursion, but it is limited to finitely many random variables.

DAPER (Heckerman *et al.*, 2004) combines the entity-relational model with DAG models to express probabilistic knowledge about structured entities and their relationships. Any model constructed in Plates or PRM can be represented by DAPER. Thus, DAPER is a unifying language for expressing relational probabilistic knowledge. DAPER expresses probabilistic models over finite databases, and cannot represent arbitrary FOPC expressions involving quantifiers. Therefore, like other languages discussed above, DAPER does not achieve full FOPC representational power.

Most of the abovementioned work is still under development, and has provided undeniable improvements in the flexibility and expressiveness of probabilistic representation. Multi-Entity Bayesian Networks, which we will cover in the next chapter, is another formal system that combines FOL and probability theory. Among the major reasons that led us to have adopted MEBN as the formal basis for PR-OWL are its ability to express joint distributions over models of arbitrary finitely axiomatizable first order theories and to add new axioms by Bayesian conditioning.



### Chapter 3 Multi-Entity Bayesian Networks

Multi-Entity Bayesian Networks integrate first order logic with Bayesian probability. MEBN logic expresses probabilistic knowledge as a collection of MEBN fragments (MFrag) organized into MEBN Theories (MTheories). An MFragment represents a conditional probability distribution of the instances of its resident random variables given the values of instances of their parents in the Fragment graphs and given the context constraints.

A collection of MFragments represents a joint probability distribution over an unbounded, possibly infinite number of instances of its random variables. The joint distribution is specified by means of the local distributions together with the conditional independence relationships implied by the fragment graphs. Context terms are used to specify constraints under which the local distributions apply.

A collection of MFragments that satisfies consistency constraints ensuring the existence of a unique joint probability distribution over its random variables is called an MTheory. MTheories can express probability distributions over truth values of arbitrary First Order Logic sequences and can be used to express domain-specific ontologies that capture statistical regularities in a particular domain of application.

In addition, MTheories can represent particular facts relevant to a given reasoning problem. Conditioning a prior distribution represented by an MTheory on its findings is the basis of probabilistic inference with MEBN logic.

Support for decision constructs in MEBN is provided via Multi-Entity Decision Graphs (MEDG), which are related to MEBN the same way influence diagrams are related to Bayesian Networks. An MEDG can be applied in any application that requires optimizing a set of alternatives (i.e. an MEDG policy) over the given constraints of a specific situation. MEBN logic also provides means of learning the structure of a MEBN Theory on the basis of data (i.e. Bayesian learning), while parameter learning can be expressed as inference in MEBN theories that contain parameter random variables.

In short, MEBN logic has the potential to serve as the mathematical backbone for future Semantic Web applications that can deal with plausible reasoning. Yet, as we will explain later in this work, some issues such as the lack of built-in support for typing and polymorphism had to be addressed before making use of the logic's strengths. In order to understand those issues and to achieve a thorough understanding of MEBN primitives, this Chapter uses the Star Trek model presented in Section 2.3 as the background to explain the principles of the logic and its representational power as well.

### 3.1 A More "Realistic" Sci-fi Scenario

The limited model of the previous section would be of little use in increasing the Captain's awareness of the level of danger faced by the *Enterprise*. In addition to the model's naïve assumptions, there were clear omissions such as the assessment of the

threat posed by a given starship, its ability and willingness to attack our own vessel, etc. These and other pertinent issues are addressed in the context of a richer scenario for which the power of MEBN is required.

Like present-day Earth, 24<sup>th</sup> Century outer space is not a politically trivial environment. It is clear that the previous model failed to consider the many different alien species with diverse profiles that populate the Universe as portrayed in the television series. Although MEBN logic can represent the full range of species inhabiting the Universe in the 24<sup>th</sup> century, for purposes of this work only a few sample groups will suffice. Therefore, in the following pages the explicitly modeled species will be restricted to Friends<sup>17</sup>, Cardassians, Romulans, and Klingons, while addressing encounters with other possible races using the general label *Unknown*.

Cardassians are constantly at war with the Federation, so any encounter with them is considered a hostile event. Fortunately, they do not possess cloaking technology, which makes it easier to detect and discriminate them. Romulans possess cloaking technology and are more ambiguous, behaving in a hostile manner in roughly half their encounters with Federation starships. Klingons, who also possess cloaking technology, have a peace agreement with the Federation of Planets, but their treacherous and aggressive behavior makes them less reliable than friends. Finally, when facing an unknown species, the historical log of such events shows that out of every ten new encounters, only one was hostile.

---

<sup>17</sup> The interested reader can find further information on the Star Trek series in a plethora of websites dedicated to preserve or to extend the history of series, such as [www.startrek.com](http://www.startrek.com), [www.ex-astris-scientia.org](http://www.ex-astris-scientia.org), or [techspecs.acalltoduty.com](http://techspecs.acalltoduty.com).

Apart from the species of its operators, a truly “realistic” model would consider each starship’s type, offensive power, the ability to inflict harm to the *Enterprise* given its range, and numerous other features pertinent to the model’s purpose.

### 3.2 The Basics of MFrag

MEBN logic represents the world as comprised of entities that have attributes and are related to other entities. Random variables (RV) represent features of entities and relationships among entities. Knowledge about attributes and relationships is expressed as a collection of MFrag organized into MTheories. An MFrag represents a conditional probability distribution for instances of its resident RVs given their parents in the fragment graph and the context nodes. An MTheory is a set of MFrag that collectively satisfies consistency constraints ensuring the existence of a unique joint probability distribution over instances of the RVs represented in each of the MFrag within the set.

Like a BN, an MFrag contains nodes, which represent RVs, arranged in a directed graph whose edges represent direct dependence relationships. An isolated MFrag can be roughly compared with a standard BN with known values for its root nodes and known local distributions for its non-root nodes. For example, the MFrag of Figure 8 represents knowledge about the degree of danger to which our own starship is exposed. The fragment graph has seven nodes. The four nodes at the top of the figure are context nodes; the two shaded rectangular nodes below the context nodes are the input nodes; and the bottom node is a resident node.

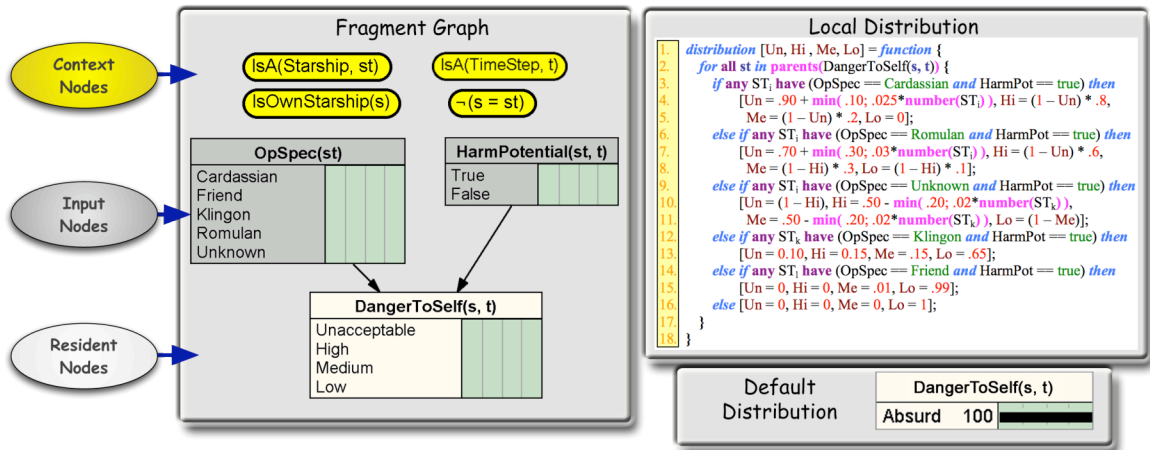


Figure 8. The Danger To Self MFragment

A node in an MFragment may have a parenthesized list of arguments. These arguments are placeholders for entities in the domain. For example, the argument  $st$  to  $HarmPotential(st, t)$  is a placeholder for an entity that has a potential to harm, while the argument  $t$  is a placeholder for the time step this instance represents. To refer to an actual entity in the domain, the argument is replaced with a *unique identifier*. By convention, unique identifiers begin with an exclamation point, and no two distinct entities can have the same unique identifier. The result of substituting unique identifiers for a RV's arguments is one or more *instances* of that RV. For example,  $HarmPotential(!ST1, !T1)$  and  $HarmPotential(!ST2, !T1)$  are two instances of  $HarmPotential(st, t)$  that both occur in the time step  $!T1$ .

The resident nodes of an MFragment have local distributions that define how their probabilities depend on the values of their parents in the fragment graph. In a complete MTheory, each random variable has exactly one *home MFragment*, where its local distribution

is defined.<sup>18</sup> Input and context nodes (e.g., *OpSpec(st)* or *IsOwnStarship(s)*) influence the distribution of the resident nodes, but their distributions are defined in their own home MFrag.

Context nodes represent conditions that must be satisfied for the influences and local distributions of the fragment graph to apply. Context nodes may have value *True*, *False*, or *Absurd*.<sup>19</sup> Context nodes having value *True* are said to be satisfied. As an example, if the unique identifier for the *Enterprise* (i.e., !ST0) is substituted for the variable *s* in *IsOwnStarship(s)*, the resulting hypothesis will be true. If, instead, a different starship unique identifier (say, !ST1) is used, then this hypothesis will be false. Finally, if the unique identifier of a non-starship (say, !Z1) replaces *s*, then this statement is absurd (i.e., it is absurd to ask whether or not a zone in space is one's own starship).

To avoid cluttering the fragment graph, the states of context nodes are not shown, contrary to what happens with input and resident nodes. This is mainly because they are Boolean nodes whose values are relevant only for deciding whether to use a resident random variable's local distribution or its default distribution.

No probability values are shown for the states of the nodes of the fragment graph in Figure 8. This is because nodes in a fragment graph do not represent individual random variables with well-defined probability distributions. Instead, a node in an MFrag

---

<sup>18</sup> Please, note that standard MEBN logic does not support polymorphism. However, an extension to a typed polymorphic version is proposed in Chapter 4, and would permit a random variable to be resident in more than one MFrag.

<sup>19</sup> State names in this Dissertation are alphanumeric strings beginning with a letter, including *True* and *False*. However, Laskey (2005) uses the symbols T for *True*, F for *False*, and  $\perp$  for *Absurd*, and requires other state names to begin with an exclamation point (because they are unique identifiers)

represents a generic class of random variables. To draw inferences or declare evidence, we must create instances of the random variable classes.

To find the probability distribution for an instance of *DangerToSelf*( $s, t$ ), the first step is to identify all instances of *HarmPotential*( $st, t$ ) and *OpSpec*( $st$ ) for which the context constraints are satisfied. If there are none, then the default distribution that assigns value *Absurd* with probability 1 must be used. Otherwise, to complete the definition of the MFrag of Figure 8, a local distribution must be specified for its lone resident node, *DangerToSelf*( $s, t$ ).

The pseudo-code of Figure 8 defines a local distribution for the danger to a starship due to all starships that influence its danger level. Local distributions in standard BNs are typically represented by static tables, which limits each node to a fixed number of parents. On the other hand, an instance of a node in an MTheory might have any number of parents. Thus, MEBN implementations (i.e. languages based on MEBN logic) must provide an expressive language for defining local distributions. In this work, the use of pseudo-code is intended to convey the idea of using local expressions to specify probability distributions, while not committing to a particular syntax.

Lines 3 to 5 cover the case in which there is at least one nearby starship operated by Cardassians and having the ability to harm the *Enterprise*. This is an uncomfortable situation for Capitan Picard, the Enterprise Commander, and his starship, where the probability of an unacceptable danger to self is 0.90 plus the minimum of 0.10 and the result of multiplying 0.025 by the total number of starships that are harmful and operated by Cardassians.

Also the remaining belief (i.e. the difference between 100% and the belief in state *Unacceptable* is divided between *High* (80% of the remainder) and *Medium* (20% of the remainder) whereas belief in *Low* is zero. The remaining lines use similar formulas to cover the other possible configurations in which there exist starships with potential to harm *Enterprise* (i.e.  $HarmPotential(st, t) = True$ ).

The last conditional statement of the local expression covers the case in which no nearby starships can inflict harm upon the *Enterprise* (i.e. all nodes  $HarmPotential(st, t)$  have value *False*). In this case, the value for  $DangerToSelf(s, t)$  is *Low* with probability 1.

Figure 9 depicts an instantiation of the Danger To Self MFragment for which there are four starships nearby, three of them operated by Cardassians and one by the Romulans. Also, the Romulan and two of the Cardassian starships are within a range at which they can harm the *Enterprise*, whereas the other Cardassian starship is too far away to inflict any harm.

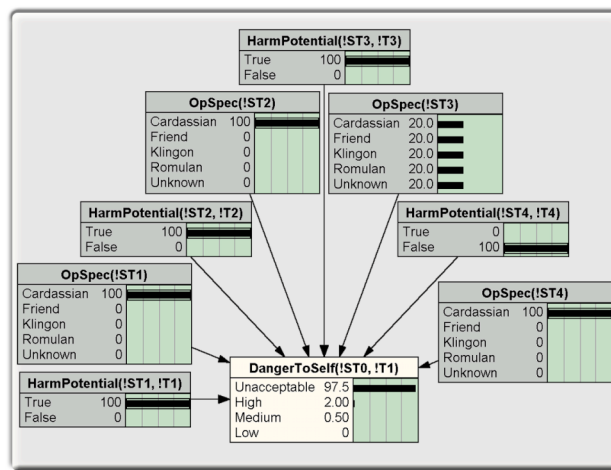


Figure 9. An Instance of the Danger To Self MFragment



Following the procedure described in Figure 8, the belief for state *Unacceptable* is .975 ( $.90 + .025*3$ ) and the beliefs for states *High*, *Medium*, and *Low* are .02 ( $((1-.975)*.8)$ ), .005 ( $((1-.975)*.2)$ ), and zero respectively.

In short, the pseudo-code covers all possible input node configurations by linking the danger level to the number of nearby starships that have the potential to harm our own starship. The formulas state that if there are any Cardassians or Romulan starships within *Enterprise's* range, then a glimpse of what would the distribution for danger level given the number of nearby starships looks like is depicted in Table 2.

Table 2. Sample Parts of the Danger To Self MFRag Probability Distribution

Relevant Starships Nearby	Danger Level Dist.
At least 1 Cardassian	[0.925, 0.024, 0.006, 0]
At least 2 Cardassians	[0.99, 0.008, 0.002, 0]
At least 3 Cardassians	[0.975, 0.2, 0.05, 0]
More than 4 Cardassians	[1, 0, 0, 0]
No Cardassians but at least 1 Romulan	[.73, .162, .081, .027]
No Cardassians but at least 1 Romulans	[.76, .144, .072, .024]
...	... (see formula)
No Cardassians but 10 or more Romulans	[1, 0, 0, 0]
No Cardassians or Romulans, one Unknown	[.02, .48, .48, .02]
...	... (see formula)
No Cardassians or Romulans, 10+ Unknown	[.20, .30, .30, .20]
...	...(see formula)

Following the same logic depicted in the formula, if there are only friendly starships nearby with the ability to harm the *Enterprise*, then the distribution becomes [0,

0, 0.01, .99]. The last line indicates that if that no starship can harm the *Enterprise*, then the danger level will be *Low* for sure.

As noted previously, a powerful formalism is needed to represent complex scenarios at a reasonable level of fidelity. In the probability distribution shown in this example, additional detail could have been added and many nuances might have been explored. For example, a large number of nearby Romulan ships might have been considered as a fair indication of a coordinated attack and therefore implied greater danger than an isolated Cardassian ship.

Nonetheless, this example was purposely kept simple in order to clarify the basic capabilities of the logic. It is clear that more complex knowledge patterns could be accommodated as needed to suit the requirements of the application. MEBN logic has built-in logical MFragments that provide the ability to express any sentence that can be expressed in first-order logic. Laskey (2005) proves that MEBN logic can implicitly express a probability distribution over interpretations of any consistent, finitely axiomatizable first-order theory. This provides MEBN with sufficient expressive power to represent virtually any scientific hypothesis.

### 3.3 Representing Recursion in MEBN Logic

One of the main limitations of BNs is their lack of support for recursion. Extensions such as dynamic Bayesian networks provide the ability to define certain kinds of recursive relationships. MEBN provides theoretically grounded support for very

general recursive definitions of local distributions. Figure 10 depicts an example of how an MFrag can represent temporal recursion.

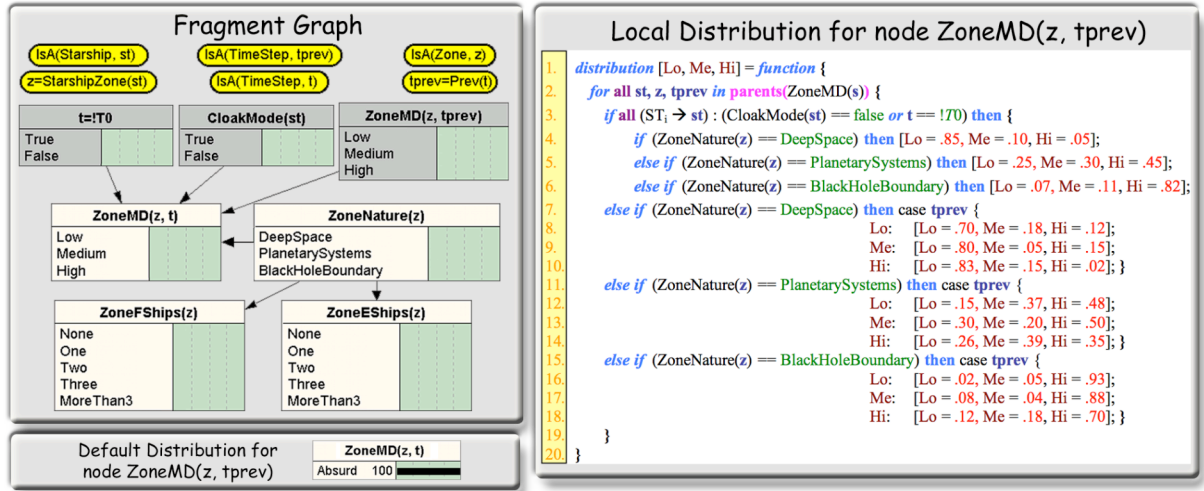


Figure 10. The Zone MFrag

In that MFrag, a careful reading of the context nodes will make it clear that in order for the local distribution to apply,  $z$  has to be a zone and  $st$  has to be a starship that has  $z$  as its current position. In addition,  $tprev$  and  $t$  must be *TimeStep* entities, and  $tprev$  is the step preceding  $t$ .

Other varieties of recursion can also be represented in MEBN logic by means of MFrams that allow influences between instances of the same random variable. Allowable recursive definitions must ensure that no random variable instance can influence its own probability distribution. General conditions that both recursive and non-recursive MFrams and MTheories must satisfy are given in Laskey (2005).

As in non-recursive MFrams, the input nodes in a recursive MFrag include nodes whose local distributions are defined in another MFrag (i.e.,  $CloakMode(st)$ ). In addition, the input nodes may include instances of recursively-defined nodes in the MFrag itself.

For example, the input node  $ZoneMD(z, t_{prev})$  represents the magnetic disturbance in zone  $z$  at the previous time step, which influences the current magnetic disturbance  $ZoneMD(z, t)$ . The recursion is grounded by specifying an initial distribution at time  $!T0$  that does not depend on a previous magnetic disturbance.

Figure 12 illustrates how recursive definitions can be applied to construct a *situation-specific Bayesian Network* (SSBN) to answer a query. In this specific case, the query concerns the magnetic disturbance at time  $!T3$  in zone  $!Z0$ , where  $!Z0$  is known to contain the uncloaked starship  $!ST0$  (*Enterprise*) and exactly one other starship  $!ST1$ , which is known to be cloaked.

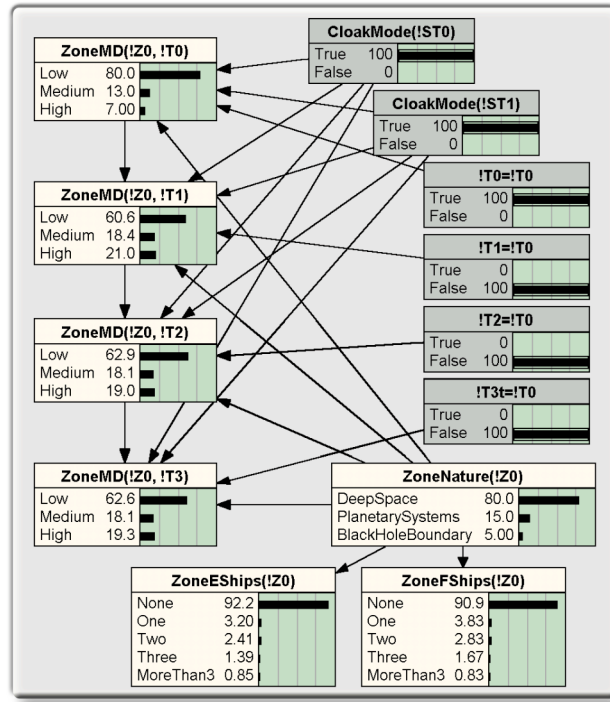


Figure 11. SSBN Constructed from Zone MFragment

The process to build the graph shown in this picture begins by creating an instance of the home MFragment of the query node  $ZoneMD(!Z0, !T3)$ . That is,  $!Z0$  is substituted for  $z$  and  $!T3$  for  $t$ , and then all instances of the remaining random variables

that meet the context constraints are created. The next step is to build any conditional probability tables (CPTs) that can already be built on the basis of the available data. CPTs for  $ZoneMD(!Z0,!T3)$ ,  $ZoneNature(!Z0)$ ,  $ZoneEShips(!Z0)$ , and  $ZoneFShips(!Z0)$  can be constructed because they are resident in the retrieved MFrag. Single-valued CPTs for  $CloakMode(!ST0)$ ,  $CloakMode(!ST1)$ , and  $!T3=!T0$  can be specified because the values of these random variables are known.

At end of the above process, only one node,  $ZoneMD(!Z0,!T2)$ , remains for which there is no CPT. To construct its CPT, its home MFrag must be retrieved, and any random variables that meet its context constraints and have not already been instantiated must be instantiated. The new random variables created in this step are  $ZoneMD(!Z0,!T1)$  and  $!T2=!T0$ . The value of the latter is already known, while the home MFrag of the former has to be retrieved. This process continues until all the nodes of Figure 11 are added. At this point, the CPTs for all random variables can be constructed, and thus the SSBN is complete.<sup>20</sup>

The MFrag depicted in Figure 10 defines the local distribution that applies to all these instances, even though for brevity only the probability distributions (local and default) for node  $ZoneMD(z, t)$  were displayed. The remaining distributions can be found in Appendix A. Note that when there is no starship with cloak mode activated, the probability distribution for magnetic disturbance given the zone nature does not change with time. When there is at least one starship with cloak mode activated, then the

---

<sup>20</sup> For efficiency reasons, most knowledge-based model construction systems would not explicitly represent root evidence nodes such as  $CloakMode(!ST0)$  or  $!T1=!T0$  or barren nodes such as  $ZoneFShips(!Z0)$  and  $ZoneFShips(!Z0)$ . For expository purposes, the approach taken here was the logically equivalent, although less computationally efficient, approach of including all these nodes explicitly.

magnetic disturbance tends to fluctuate regularly with time in the manner described by the local expression. For the sake of simplicity, the underlying assumption that the local distribution depends only on whether there is a cloaked starship nearby was adopted, although in a more “realistic” model the disturbance might increase with the number of cloaked starships and/or the power of the cloaking device.

Another implicit assumption taken in this example regards the initial distribution for the magnetic disturbance when there are cloaked starships, which was assumed to be equal to the stationary distribution given the zone nature and the number of cloaked starships present initially. Of course, it would be possible to write different local expressions expressing a dependence on the number of starships, their size, their distance from the *Enterprise*, etc.

MFragments provide a flexible means to represent knowledge about specific subjects within the domain of discourse, but the true gain in expressive power is revealed when these “knowledge patterns” are aggregated in order to form a coherent model of the domain of discourse that can be instantiated to reason about specific situations and refined through learning.

It is important to note that just collecting a set MFragments that represent specific parts of a domain is not enough to ensure a coherent representation of that domain. For example, it would be easy to specify a set of MFragments with cyclic influences, or one having multiple conflicting distributions for a random variable in different MFragments. The following section describes how to define complete and coherent domain models as collections of MFragments.

### 3.4 Building MEBN Models with MTheories

In order to build a coherent model, it is mandatory that the set of MFragments collectively satisfies consistency constraints ensuring the existence of a unique joint probability distribution over instances of the random variables mentioned in the MFragments. Such a coherent collection of MFragments is called an MTheory.

An MTheory represents a joint probability distribution for an unbounded, possibly infinite number of instances of its random variables. This joint distribution is specified by the local and default distributions within each MFragment together with the conditional independence relationships implied by the fragment graphs.

The MFragments described above are part of a *generative MTheory* for the intergalactic conflict domain. A generative MTheory summarizes statistical regularities that characterize a domain. These regularities are captured and encoded in a knowledge base using some combination of expert judgment and learning from observation.

To apply a generative MTheory to reason about particular scenarios, it is necessary to provide the system with specific information about the individual entity instances involved in the scenario. On receipt of this information, Bayesian inference can be used both to answer specific questions of interest (e.g., how high is the current level of danger to the *Enterprise*?) and to refine the MTheory (e.g., each encounter with a new species gives additional statistical data about the level of danger to the *Enterprise* from a starship operated by an unknown species). Bayesian inference is used to perform both problem-specific inference and learning in a sound, logically coherent manner.

*Findings* are the basic mechanism for incorporating observations into MTheories. A finding is represented as a special 2-node MFrag containing a node from the generative MTheory and a node declaring one of its states to have a given value. From a logical point of view, inserting a finding into an MTheory corresponds to asserting a new axiom in a first-order theory. In other words, MEBN logic is inherently open, having the ability to incorporate new axioms as evidence and update the probabilities of all random variables in a logically consistent way.

In addition to the requirement that each random variable must have a unique home MFrag, a valid MTheory must ensure that all recursive definitions terminate in finitely many steps and contain no circular influences. Finally, as demonstrated above, random variable instances may have a large, and possibly unbounded number of parents.

A valid MTheory must satisfy an additional condition to ensure that the local distributions have reasonable limiting behavior as more and more parents are added. Laskey (2005) proved that when an MTheory satisfies these conditions (as well as other technical conditions that are unimportant to our example), then there exists a joint probability distribution on the set of instances of its random variables that is consistent with the local distributions assigned within its MFrag.

Furthermore, any consistent, finitely axiomatizable FOL theory can be translated to infinitely many MTheories, all having the same purely logical consequences, that assign different probabilities to statements whose truth-value is not determined by the axioms of the FOL theory.



MEBN logic contains a set of built-in logical MFrag (including quantifier, indirect reference, and Boolean connective MFrag) that provide the ability to represent any sentence in first-order logic. If the MTheory satisfies additional conditions, then a conditional distribution exists given any finite sequence of findings that does not logically contradict the logical constraints of the generative MTheory. MEBN logic thus provides a logical foundation for systems that reason in an open world and incorporate observed evidence in a mathematically sound, logically coherent manner.

Figure 12 shows an example of a generative MTheory for the *Star Trek* domain. For the sake of conciseness, the local distribution formulas and the default distributions are not shown here.

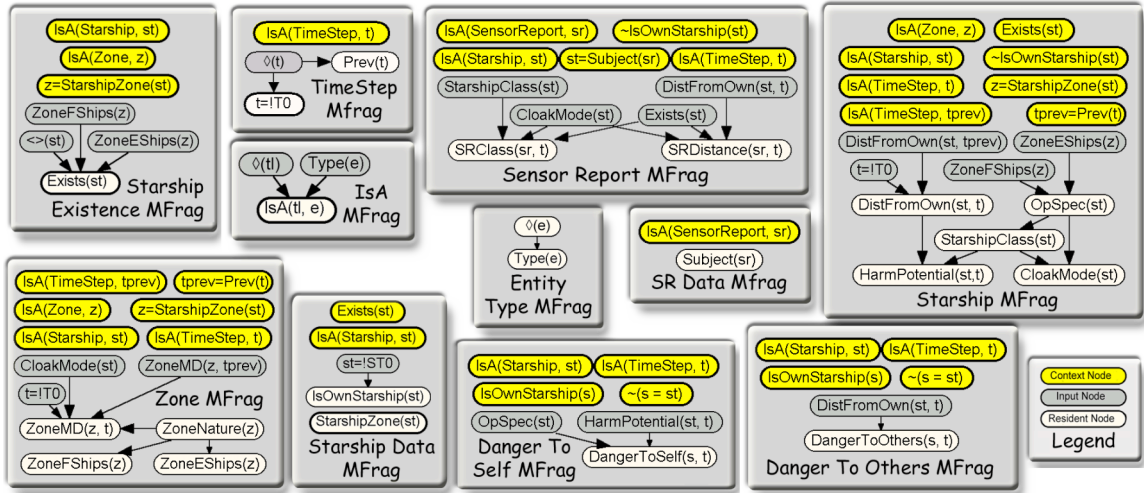


Figure 12. The Star Trek Generative MTheory

The Entity Type MFrag, at the right side of Figure 12, is meant to formally declare the possible types of entity that can be found in the model. This is a generic MFrag that allows the creation of domain-oriented types (which are represented by TypeLabel entities). This MFrag forms the basis for a Type system.

The simple model depicted here did not address the creation or the explicit support for entity types. Standard MEBN logic as defined in Laskey (2005) is untyped, meaning that a knowledge engineer who wishes to represent types must explicitly define the necessary logical machinery. Typing is an important feature in ontology languages such as OWL, so as part of this research effort we have developed an extended version of the MEBN logic that includes built-in support for typing. This extension is explained in Chapter 4.

The Entity Type MFrag of Figure 12 defines an extremely simple kind of type structure. MEBN can be extended with MFrams to accommodate any flavor of type system, including more complex capabilities such as sub-typing, polymorphism, multiple-inheritance, etc.

It is important to understand the power and flexibility that MEBN logic gives to knowledge base designers by allowing multiple, equivalent ways of portraying the same knowledge. Indeed, the generative MTheory of Figure 12 is just one of the many possible (consistent) sets of MFrams that can be used to represent a given joint distribution. There, the random variables were clustered in a way that attempts to naturally reflect the structure of the objects in that scenario (i.e. an object oriented approach to modeling was taken), but this was only one design option among the many allowed by the logic.

As an example of such flexibility, Figure 13 depicts the same knowledge contained in the Starship MFrag of Figure 12 (right side) using three different MFrams. In this case, the modeler might have opted for decomposing an MFrag in order to get the extra flexibility of smaller, more specific MFrams that can be combined in different ways.

Another knowledge engineer might prefer the more concise approach of having all knowledge in just one MFrag. Ultimately, the approach to be taken when building an MTheory will depend on many factors, including the model's purpose, the background and preferences of the model's stakeholders, the need to interface with external systems, etc.

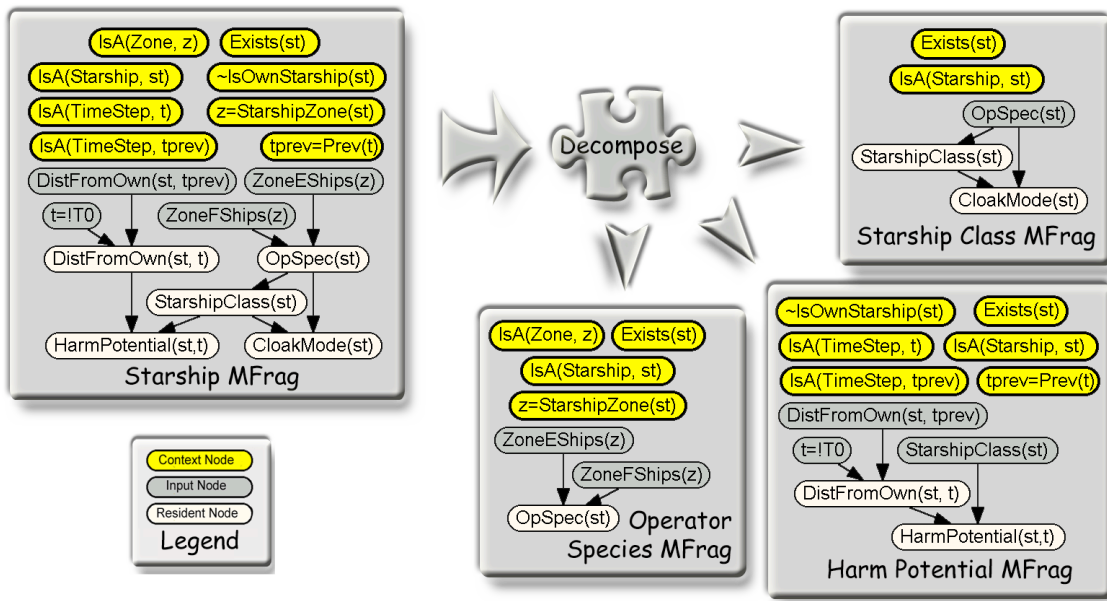


Figure 13. Equivalent MFrag Representations of Knowledge

First Order Logic (or one of its subsets) provides the theoretical foundation for the type systems used in popular object-oriented and relational languages. MEBN logic provides the basis for extending the capability of these systems by introducing a sound mathematical basis for representing and reasoning under uncertainty, which is precisely the idea being explored in the extensions that will be proposed in the next Chapter. The advantages of a MEBN-based type system are also explored in that Chapter.

Another powerful aspect of MEBN, the ability to support *finite or countably infinite recursion*, is illustrated in the Sensor Report and Zone MFrag, both of which

involve temporal recursion. The Time Step MFrag includes a formal specification of the local distribution for the initial step of the time recursion (i.e. when  $t=!T0$ ) and of its recursive steps (i.e. when  $t$  does not refer to the initial step). Other kinds of recursion can be represented in a similar manner.

MEBN logic also has the ability to represent and reason about hypothetical entities. Uncertainty about whether a hypothesized entity actually exists is called *existence uncertainty*. In the example model presented here, the random variable *Exists(st)* is used to reason about whether its argument is an actual starship. For example, it might be uncertain whether a sensor report corresponds to one of the starships already known by the system, a starship of which the system was not previously aware of, or a spurious sensor report.

To allow for hypothetical starships, the local distribution for *Exists(st)* assigns non-zero probability to *False*. Suppose the unique identifier *!ST4* refers to a hypothetical starship nominated to explain the report. In this case, *IsA(Starship, !ST4)* has value *True*, but the value of *Exists(!ST4)* is uncertain. A value of *False* would mean *!ST4* is a spurious starship or false alarm. Queries involving the unique identifier of a hypothetical starship return results weighted by our belief that it is an actual or a spurious starship. Belief in *Exists(!ST4)* is updated by Bayesian conditioning as relevant evidence accrues. Representing existence uncertainty is particularly useful for counterfactual reasoning and reasoning about causality (Druzdzel & Simon, 1993; Pearl, 2000).

Because the *Star Trek* model was designed to demonstrate the capabilities of MEBN logic, the approach taken was to avoid issues that could be handled by the logic

but would make the model too complex. As an example, one aspect that this model does not consider is *association uncertainty*, a very common problem in multi-sensor data fusion systems. Association uncertainty means we are not sure about the source of a given report. For example, we may receive a report, !SR4, indicating a starship near a given location. Suppose we cannot tell whether the report was generated by !ST1 or !ST3, two starships known to be near the reported location, or by a previously unreported starship !ST4. In this case, we would enumerate these three unique identifiers as possible values for *Subject*(!SR4), and specify that *Exists*(!ST4) has value *False* if *Subject*(!SR4) has any value other than !ST4. Many weakly discriminatory reports coming from possibly many starships produces an exponential set of combinations that require special *hypothesis management* methods (Stone *et al.*, 1999).

Closely related to association uncertainty is *identity uncertainty*, or uncertainty about whether two expressions refer to the same entity. Association uncertainty can be regarded as a special case of identity uncertainty – that is, uncertainty about the identity of *Subject*(!SR4). The ability to represent existence, association, and identity uncertainty provides a logical foundation for hypothesis management in multi-source fusion.

The *Star Trek* model was built in a way to avoid these problems by assuming that the *Enterprise*'s sensor suite can achieve perfect discrimination. However, the underlying logic can represent and reason with association, existence, and type uncertainty, and thus provides a sound logical foundation for hypothesis management in multi-source fusion.

### 3.5 Making Decisions with Multi-Entity Decision Graphs.

Captain Picard has more than an academic interest in the danger from nearby starships. He must make decisions with life and death consequences. Multi-Entity Decision Graphs (MEDGs, or “medges”) extend MEBN logic to support decision making under uncertainty. MEDGs are related to MEBNs in the same way influence diagrams are related to Bayesian Networks. A MEDG can be applied to any problem that involves optimal choice from a set of alternatives subject to given constraints.

When a decision M<sub>Frag</sub> (i.e. one that has decision and utility nodes) is added to a generative M<sub>Theory</sub> such as the one portrayed in Figure 12, the result is a MEDG. As an example, Figure 14 depicts a decision M<sub>Frag</sub> representing Captain Picard’s choice of which defensive action to take. The decision node *DefenseAction(s)* represents the set of defensive actions available to the Captain (in this case, to fire the ship’s weapons, to retreat, or to do nothing). The value nodes capture Picard’s objectives, which in this case are to protect the *Enterprise* while also avoiding harm to innocent people as a consequence of his defensive actions. Both objectives depend upon Picard’s decision, while *ProtectSelf(s)* is influenced by the perceived danger to *Enterprise* and *ProtectOthers(s)* is depends on the level of danger to other starships in the vicinity.

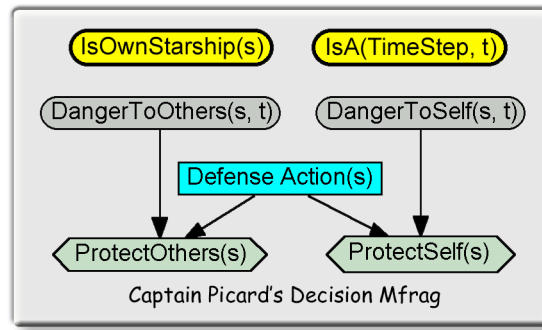


Figure 14. The Star Trek Decision Mfrag

The model described here is clearly an oversimplification of any “real” scenario a Captain would face. Its purpose is to convey the core idea of extending MEBN logic to support decision-making. Indeed, a more common situation is to have multiple, mutually influencing, often conflicting factors that together form a very complex decision problem, and require trading off different attributes of value. For example, a decision to attack would mean that little power would be left for the defense shields; a retreat would require aborting a very important mission.

MEDGs provide the necessary foundation to address all the above issues. Readers familiar with influence diagrams will appreciate that the main concepts required for a first-order extension of decision theory are all present in Figure 14. In other words, MEDGs have the same core functionality and characteristics of common MFrag. Thus, the utility table in *Survivability(s)* refers to the entity whose unique identifier substitutes for the variable  $s$ , which according to the context nodes should be our own starship (*Enterprise* in this case). Likewise, the states of input node *DangerToSelf(s, t)* and the decision options listed in *DefenseAction(s)* should also refer to the same entity.

Of course, this confers to MEDGs the expressive power of MEBN models, which includes the ability to use this same decision Mfrag to model the decision process of the

Captain of another starship. Notice that a MEDG Theory should also comply with the same consistency rules of standard MTheories, along with additional rules required for influence diagrams (e.g., value nodes are deterministic and must be leaf nodes or have only value nodes as children).

In the present example, adding the Star Trek Decision MFrag of Figure 14 to the generative MTheory of Figure 12 will maintain the consistency of the latter, and therefore the result will be a valid generative MEDG Theory. That simple illustration can be extended to more elaborate decision constructions, providing the flexibility to model decision problems in many different applications spanning diverse domains.

### 3.6 Inference in MEBN Logic.

A generative MTheory provides prior knowledge that can be updated upon receipt of evidence represented as finding MFrag. We now describe the process used to obtain posterior knowledge from a generative MTheory and a set of findings.

In a BN model such as the ones shown from Figure 5 through Figure 7, assessing the impact of new evidence involves conditioning on the values of evidence nodes and applying a belief propagation algorithm. When the algorithm terminates, beliefs of all nodes, including the node(s) of interest, reflect the impact of all evidence entered thus far. This process of entering evidence, propagating beliefs, and inspecting the posterior beliefs of one or more nodes of interest is called a query.

MEBN inference works in a similar way (after all, MEBN is a Bayesian logic), but following a more complex yet more flexible process. Whereas BNs are static models



that must be changed whenever the situation changes (e.g. number of starships, time recursion, etc.), an MTheory implicitly represents an infinity of possible scenarios. In other words, the MTheory represented in Figure 12 (as well as the MEDG obtained by aggregating the MFrag in Figure 14) is a model that can be used for as many starships as wanted, and for as many time steps that are necessary to get the conclusions needed.

That said, the obvious question is how to perform queries within such a model. A simple example of query processing was given above in Section 3.3. Here, the general algorithm for constructing a situation-specific Bayesian network (SSBN) is described in a general way. In order to execute such algorithm, it is necessary to have an initial generative MTheory (or MEDG Theory), a Finding set (which conveys particular information about the situation) and a Target set (which indicates the nodes of interest to the query being made).

For comparison, let's suppose there is a situation similar to the one in Figure 3, where four starships are within the *Enterprise*'s range. In that particular case, a BN was used to represent the situation at hand, which means the model is "hardwired" to a known number (four) of starships, and any other number would require a different model. A standard Bayesian inference algorithm applied to that model would involve entering the available information about these four starships (i.e., the four sensor reports), propagating the beliefs, and obtaining posterior probabilities for the hypotheses of interest (e.g., the four *Starship Type* nodes).

Similarly, MEBN inference begins when a query is posed to assess the degree of belief in a target random variable given a set of evidence random variables. We start

with a generative MTheory, add a set of finding MFragS representing problem-specific information, and specify the target nodes for our query. The first step in MEBN inference is to construct the SSBN, which can be seen as an ordinary Bayesian network constructed by creating and combining instances of the MFragS in the generative MTheory.

Next, a standard Bayesian network inference algorithm is applied. Finally, the answer to the query is obtained by inspecting the posterior probabilities of the target nodes. A MEBN inference algorithm is provided in Laskey (2005). The algorithm presented there does not handle decision graphs. Thus, the illustration presented in the following lines extends the algorithm for purposes of demonstrating how the MEDG Theory portrayed in Figure 12 and Figure 14 can be used to support the Captain's decision.

In this example, the finding MFragS convey information that there are five starships (!ST0 through !ST4) and that the first is *Enterprise* itself. For the sake of illustration, let's assume that the Finding set also includes data regarding the nature of the space zone *Enterprise* is currently located (!Z0), its magnetic disturbance for the first time step (!T0), and sensor reports for starships !SR1 to !SR4 for the first two time steps.

Let's also assume that the Target set for this illustrative query includes an assessment of the level of danger experienced by the *Enterprise* and the best decision to take given this level of danger.

Figure 15 shows the situation-specific Bayesian network for such query<sup>21</sup>. To construct that SSBN, the initial step is to create instances of the random variables in the Target set and the random variables for which there are findings. The target random variables are *DangerLevel(!ST0)* and *DefenseAction(!ST0)*. The finding random variables are the eight *SRDistance* nodes (2 time steps for each of four starships) and the two *ZoneMD* reports (one for each time step). Although each finding MFrags contains two nodes, the random variable on which there is a finding and a node indicating the value to which it is set, only the first of these is included in our situation-specific Bayesian network, and declared as evidence that its value is equal to the observed value indicated in the finding MFrags. Evidence nodes are shown with bold borders.

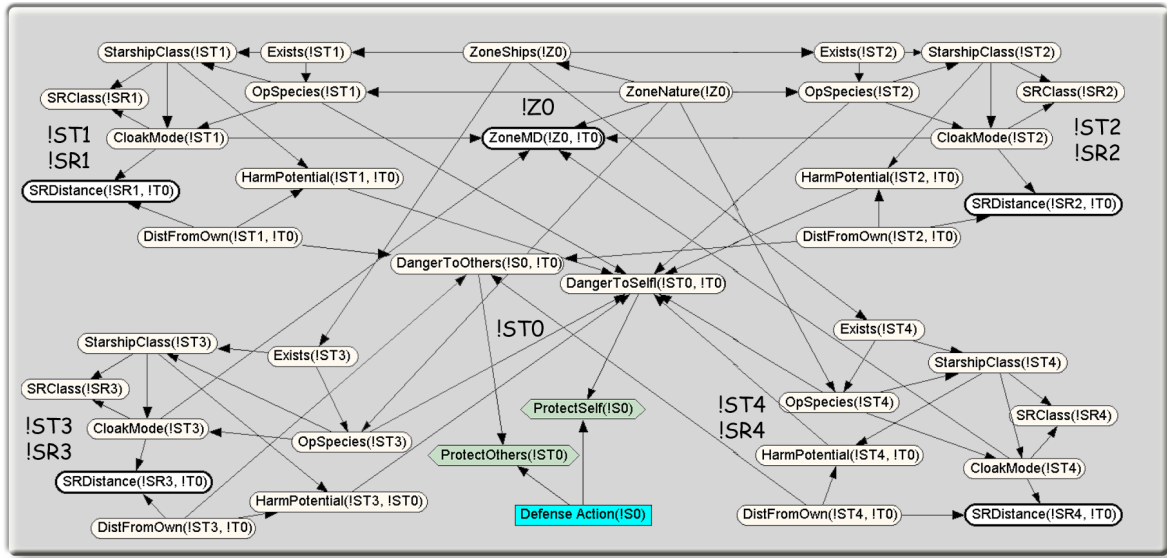


Figure 15. SSBN for the Star Trek MTheory with Four Starships within Range

The next step is to retrieve and instantiate the home MFrags of the finding and target random variables. When each MFrags is instantiated, instances of its random

<sup>21</sup> The alert reader may notice that root evidence nodes and barren nodes that were included in the constructed network of Figure 8 are not included here. As noted above, explicitly representing these nodes is not necessary.

variables are created to represent known background information, observed evidence, and queries of interest to the decision maker. If there are any random variables with undefined distributions, then the algorithm proceeds by instantiating their respective home MFrag.

The process of retrieving and instantiating MFrag continues until there are no remaining random variables having either undefined distributions or unknown values. The result, if this process terminates, is the *SSBN* or, in this example, a *situation-specific decision graph* (SSDG).

In some cases the SSBN can be infinite, but under conditions given in Laskey (Laskey, 2005), the algorithm produces a sequence of approximate SSBNs for which the posterior distribution of the target nodes converges to their posterior distribution given the findings. Mahoney and Laskey (1998) define a SSBN as a minimal Bayesian network sufficient to compute the response to a query. A SSBN may contain any number of instances of each MFrag, depending on the number of entities and their interrelationships. The SSDG in Figure 15 is the result of applying this process to the MEDG Theory obtained with the aggregation of Figure 12 and Figure 14 with the Finding and Target set defined above.

Another important use for the SSBN algorithm is to help in the task of performing Bayesian learning, which is treated in MEBN logic as a sequence of MTheories.

### 3.7 Learning from Data.

Learning graphical models from observations is usually divided into two different categories inferring the parameters of the local distributions when the structure is known, and inferring the structure itself. In MEBN, by structure we mean the possible values of the random variables, their organization into MFrag, the fragment graphs, and the functional forms of the local distributions.

Figure 16 shows an example of parameter learning in MEBN logic in which we adopt the assumption that one can infer the length of a starship on the basis of the average length of all starships. This generic domain knowledge is captured by the generative MFrag, which specifies a prior distribution based on what we know about starship lengths.

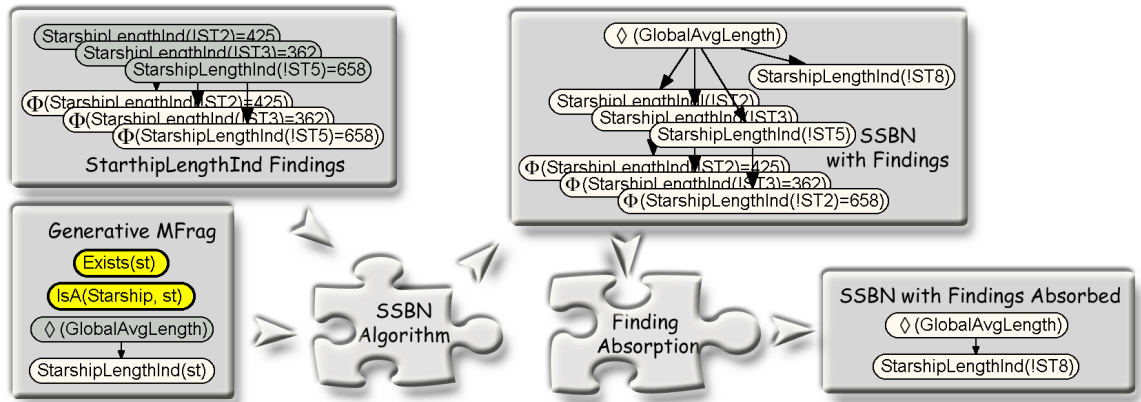


Figure 16. Parameter Learning in MEBN

One strong point about using Bayesian models in general and MEBN logic in particular is the ability to refine prior knowledge as new information becomes available. In our example, let's suppose that the Enterprise system receives precise information on

the length of starships !ST2, !ST3, and !ST5; but has no information regarding the incoming starship !ST8.

The first step of this simple parameter learning example is to enter the available information to the model in the form of findings (see box StarshipLengthInd Findings). Then, a query is posed on the length of !ST8. The SSBN algorithm will instantiate all the random variables that are related to the query at hand until it finishes with the SSBN depicted in Figure 16 (box SSBN with Findings).

In this example, the MFragS satisfy graph-theoretic conditions under which a restructuring operation called *finding absorption* (Buntine, 1994b) can be applied. Therefore, the prior distribution of the random variable *GlobalAvgLength* can be replaced in the SSBN by the posterior distribution obtained after adding evidence in the form of findings<sup>22</sup>.

As a result of this learning process, the probability distribution for *GlobalAvgLength* has been refined in light of the new information conveyed by the findings. The resulting, more precise distribution can now be used not only to predict the length of !ST8 but for future queries as well. In this specific example, the same query would retrieve the SSBN in the lower right corner of Figure 16 (box SSBN with Findings Absorbed).

One of the major advantages of the finding absorption operation is that it greatly improves the tractability of both learning and SSBN inference. Finding absorption can

---

<sup>22</sup> Absorption changes the structure of the already-observed length MFragS by removing their dependence on the global average length and setting their observed values to probability 1. It also removes the finding MFragS for these random variables.

also be applied to modify the generative MFragS themselves, thus creating a new generative MTheory that has the same conditional distribution given its findings as the original MTheory. In this new MTheory, the distribution of *GlobalAvgLength* has been modified to incorporate the observations and the finding random variables are set with probability 1 to their observed values. Restructuring MTheories via finding absorption can increase the efficiency of SSBN construction and of inference.

Structure learning in MEBN works in a similar fashion. As an example, let's suppose that when analyzing the data that was acquired in the parameter learning process above, a domain expert raises the hypothesis that the length of a given starship might depend on its class. To put it into a "real-life" perspective, let's consider two classes: Explorers and Warbirds. The first usually are vessels crafted for long distance journeys with a relatively small crew and payload. Warbirds, on the other hand, are heavily armed vessels designed to be flagships of a combatant fleet, usually carrying lots of ammunition, equipped with many advanced technology systems and a large crew. Therefore, our expert thinks it likely that the average length of Warbirds may be greater than the average length of Explorers.

In short, the general idea of this simple example is to mimic the more general situation in which we have a potential link between two attributes (i.e. starship length and class) but at best weak evidence to support the hypothesized correlation. This is a typical situation in which Bayesian models can use incoming data to learn both structure and parameters of a domain model. Generally speaking, the solution for this class of

situations is to build two different structures and apply Bayesian inference to evaluate which structure is more consistent with the data as it becomes available.

The initial setup of the structure learning process for this specific problem is depicted in Figure 17. Each of the two possible structures is represented by its own generative MFrag. The first MFrag is the same as before: the length of a starship depends only on a global average length that applies to starships of all classes. The upper left MFrag of Figure 17, *StarshipLengthInd* MFrag conveys this hypothesis. The second possible structure, represented by the *ClassAvgLength* and *StarshipLengthDep* MFrams, covers the case in which a starship class influences its length.

The two structures are then connected by the *Starship Length* MFrag, which has the format of a *multiplexor* MFrag. The distribution of a multiplexor node such as *StarshipLength(st)* always has one parent *selector* node defining which of the other parents is influencing the distribution in a given situation.

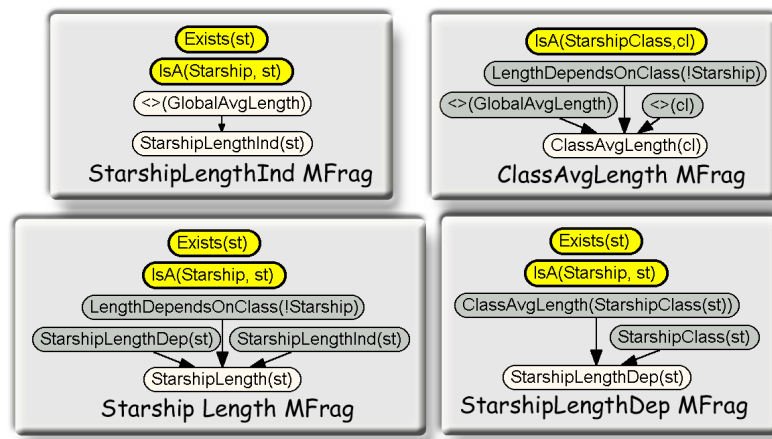


Figure 17. Structure Learning in MEBN

In this example, where there are only two possible structures, the selector parent will be a two-state node. Here, the selector parent is the Boolean



*LengthDependsOnClass(!Starship)*. When this node has value *False* then *StarshipLength(cl)* will be equal to *StarshipLengthInd(st)*, the distribution of which does not depend on the starship's class. Conversely, if the selector parent has value *True* then *StarshipLength(cl)* will be equal to *StarshipLengthDep(st)*, which is directly influenced by *ClassAvgLength(StarshipClass(st))*.

Figure 18 shows the result of applying the SSBN algorithm to the generative MFrag in Figure 17. The SSBN on the left does not have the findings included, but only information about the existence of four starships. It can be noted that the prior chosen for the selector parent (the Boolean node on the top of the SSBN) was the uniform distribution, which means that both structures (i.e. class affecting length or not) have the same prior probability.

The SSBN in the right side considers the known facts that *!ST2* and *!ST3* belong to the class of starships *!Explorer*, and that *!ST5* and *!ST8* are Warbird vessels. Further, the lengths of three ships for which there are reliable reports were also considered. The result of the inference process was not only an estimate of the length of *!ST8* but a clear confirmation that the data available strongly supports the hypothesis that the class of a starship influences its length.

It may seem cumbersome to define different random variables, *StarshipLengthInd* and *StarshipLengthDep*, for each hypothesis about the influences on a starship's length. As the number of structural hypotheses becomes large, this can become quite unwieldy. Fortunately, this difficulty can be circumvented by introducing a typed version of MEBN and allowing the distributions of random variables to depend on the type of their

argument. A detailed presentation of typed MEBN, which also extends the standard specification to allow polymorphism is the subject of the next Chapter.

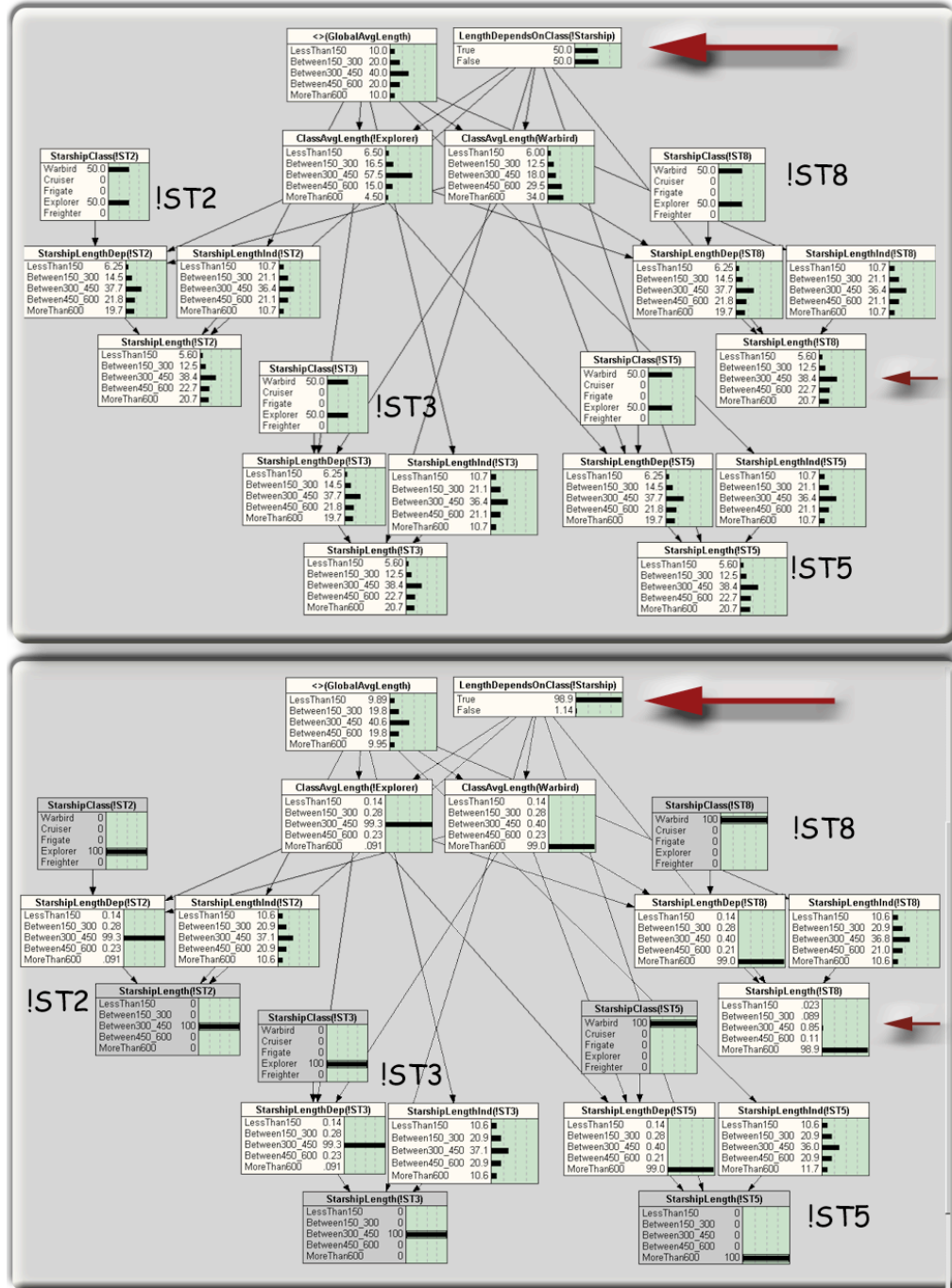


Figure 18. SSBNs for the Parameter Learning Example

This basic construction is compatible with the standard approaches to Bayesian structure learning in graphical models (e.g. Cooper & Herskovits, 1992; Heckerman *et al.*, 1995a; Jordan, 1999; Friedman & Koller, 2000)

For a detailed account of the SSBN construction algorithm and Bayesian learning with MEBN logic, the interested reader should refer to Laskey (2005). There, it is possible to find the mathematical explanation and respective logical proof for the many intricate possibilities when instantiating MFrag, such as nodes with an infinite number of states, situations where we face the prospect of large finite or countably infinite recursions, what happens when the algorithm is started with an inconsistent MTheory, etc. Also, the text provides a detailed account of how to represent any First Order Logic sentence as an MFrag using Skolem variables and quantifiers. These issues go beyond the scope of this work, since the information already covered up to this point is enough for the purposes of understanding and using Multi-Bayesian Networks as the framework for extending a web language to Bayesian first-order logic expressivity. Yet, before entering the next Chapter, it is necessary to make a brief visit to the semantics of MEBN logic, understand why it is a Bayesian first-order logic, and to address its relationship with classical logic and other formalisms as well.

### 3.8 MEBN Semantics.

In classical logic, the most that can be said about a hypothesis that can be neither proven nor disproven is that its truth-value is unknown. Practical reasoning demands more. Captain Picard's life depends on assessing the plausibility of many hypotheses he

can neither prove nor disprove. Yet, he also needs first-order logic's ability to express generalizations about properties of and relationships among entities. In short, he needs a probabilistic logic with first-order expressive power.

Although there have been many attempts to integrate classical first-order logic with probability (see discussion on Section 2.5), MEBN is the first fully first-order Bayesian logic (Laskey, 2005). MEBN logic can assign probabilities in a logically coherent manner to any set of sentences in first-order logic, and can assign a conditional probability distribution given any consistent set of finitely many first-order sentences. That is, anything that can be expressed in first-order logic can be assigned a probability by MEBN logic. The probability distribution represented by an MTheory can be updated via Bayesian conditioning to incorporate any finite sequence of findings that are consistent with the MTheory and can be expressed as sentences in first-order logic. If findings contradict the logical content of the MTheory, this can be discovered in finitely many steps. Although exact inference may not be possible for some queries, if SSBN construction will converge to the correct result if one exists.

Semantics in classical logic is typically defined in terms of possible worlds. Each possible world assigns values to random variables<sup>23</sup> in a manner consistent with the theory's axioms. For example, in the scenario illustrated in Figure 11, every possible world must assign value *True* to *CloakMode(!ST1)* and *!Z0* to *StarshipZone(!ST0)* (the latter is not explicitly represented in the figure). The value of the random variable *Zone-*

---

<sup>23</sup> In classical logic, the terms *predicate* and *function* are used in place of Boolean and non-Boolean random variables, respectively. Predicates must have value *True* or *False*, and cannot have value *Absurd*.

*Nature(!Z0)* must be one of *DeepSpace*, *PlanetarySystems*, or *BlackHoleBoundary*, but subject to that constraint, it may have different values in different possible worlds.

In classical logic, inferences are valid if the conclusion is true in all possible worlds in which the premises are true. For example, classical logic allows us to infer that *Prev(Prev(!ST4))* has value *!ST2* from the information that *Prev(!ST4)* has value *!ST3* and *Prev(!ST3)* has value *!ST2*, because the first statement is true in all possible worlds in which the latter two statements are true. But in the scenario above, classical logic permits us to draw no conclusions about the value of *ZoneNature(!Z0)* except that it is one of the three values *DeepSpace*, *PlanetarySystems*, or *BlackHoleBoundary*.

An MTheory assigns probabilities to sets of worlds. This is done in a way that ensures that the set of worlds consistent with the logical content of the MTheory has probability 100%. Each random variable instance maps a possible world to the value of the random variable in that world. In statistics, random variables are defined as functions mapping a sample space to an outcome set. For MEBN random variable instances, the sample space is the set of possible worlds. For example, *ZoneNature(!Z0)* maps a possible world to the nature of the zone labeled *!Z0* in that world. The probability that *!Z0* is a deep space zone is the total probability of the set of possible worlds for which *ZoneNature(!Z0)* has value *DeepSpace*.

In any given possible world, the generic random variable class *ZoneNature(z)* maps its argument to the nature of the zone whose identifier was substituted for the argument *z*. Thus, the sample space for the random variable class *ZoneNature(z)* is the set of unique identifiers that can be substituted for the argument *z*. Information about

statistical regularities among zones is represented by the local distributions of the MFragS whose arguments are zones. As stated in section 3.7, MFragS for parameter and structure learning provide a means for using observed information about zones to make better predictions about zones were not yet seen.

As more information is obtained about which possible world might be the actual world, the probabilities of all related properties of the world must be adjusted in a logically coherent manner. This is accomplished by adding findings to an MTheory to represent the new information, and then using Bayesian conditioning to update the probability distribution represented by the revised MTheory.

For example, suppose the system receives confirmed information that at least one enemy ship is navigating in !Z0. This information means that worlds in which *ZoneEShips(!Z0)* has value *Zero* are no longer possible. In classical logic, this new information makes no difference to the inferences one can draw about *ZoneNature(!Z0)*. All three values were possible before that new information arrived (i.e. there's at least one enemy starship in !Z0 for sure), and all three values remain possible. The situation is different in a probabilistic logic. To revise the current probabilities, it is necessary to first assign probability zero to the set of worlds in which !Z0 contains no enemy ships. Then, the probabilities of the remaining worlds should be divided by the prior probability that *ZoneEShips(!Z0)* had a value other than *Zero*. This ensures that the set of worlds consistent with the new knowledge has probability 100%. These operations can be accomplished in a computationally efficient manner using SSBN construction.

Just as in classical logic, all three values of *ZoneEShips(!Z0)* remain possible. However, their probabilities are different from their previous values. Because deep space zones are more likely than other zones to contain no ships, more of the probability in the discarded worlds was assigned to worlds in which !Z0 was a deep space zone than to worlds in which !Z0 was not in deep space. Worlds that remain possible tended to put more probability on planetary systems and black hole boundaries than on deep space. The result is a substantial reduction in the probability that !Z0 is in deep space.

Achieving full first-order expressive power in a Bayesian logic is non-trivial. This requires the ability to represent an unbounded or possibly infinite number of random variables, some of which may have an unbounded or possibly infinite number of possible values. We also need to be able to represent recursive definitions and random variables that may have an unbounded or possibly infinite number of parents. Random variables taking values in uncountable sets such as the real numbers present additional difficulties. Details on how MEBN handles these subtle issues are provided by Laskey (2005).

To our knowledge, the formulation of MEBN logic provided in Laskey (2005) is the first probabilistic logic to possess all of the following properties: (1) the ability to express a globally consistent joint distribution over models of any consistent, finitely axiomatizable FOL theory; (2) a proof theory capable of identifying inconsistent theories in finitely many steps and converging to correct responses to probabilistic queries; and (3) built in mechanisms for refining theories in the light of observations in a mathematically sound, logically coherent manner.

As such, MEBN should be seen not as a competitor, but as a logical foundation for the many emerging languages that extend the expressive power of standard Bayesian networks and/or extend a subset of first-order logic to incorporate probability.

MEBN logic brings together two different areas of research: probabilistic reasoning and classical logic. The ability to perform plausible reasoning with the expressiveness of First-Order Logic opens the possibility to address problems of greater complexity than heretofore possible in a wide variety of application domains.

XML-based languages such as RDF and OWL are currently being developed using subsets of FOL. MEBN logic can provide a logical foundation for extensions that support plausible reasoning. This work is geared towards that end, and the language proposed here, PR-OWL, is a MEBN-based extension to the SW language OWL.

The main objective of such extension is to create a language capable of representing and reasoning with probabilistic ontologies. This technology would facilitate the development of “probability-friendly” applications for the Semantic Web. The ability to handle uncertainty is clearly needed, because the SW is an open environment where uncertainty is the rule.

Probabilistic ontologies are also a very promising technique for addressing the semantic mapping problem, a difficult task whose applications range from automatic Semantic Web agents, which must be able to deal with multiple, diverse ontologies, to automated decision systems, which usually have to interact and reason with many legacy systems, each having its own distinct rules, assumptions, and terminologies.



MEBN is still in its infancy as a logic, but has already shown the potential to provide the necessary mathematical foundation for plausible reasoning in an open world characterized by many interacting entities related to each other in diverse ways and having many uncertain features and relationships. In order to realize that potential, the first step is to extend the logic so it can handle complex features that are required in expressive languages such as OWL. This is the core objective of the next Chapter.

## Chapter 4 The Path to Probabilistic Ontologies

Representing and reasoning under uncertainty is a necessary step for realizing the W3C's vision for the Semantic Web. The title of this Dissertation leaves no questions about our understanding that such step has to be taken via Bayesian probability theory, which not only allows for a principled representation of uncertainty but also provides both a proof theory for combining prior knowledge with observations, and a learning theory for refining the ontology as evidence accrues.

A key concept for achieving that goal is the one of probabilistic ontologies, so we begin by defining what we mean when using this term. Intuitively, an ontology that has probabilities attached to some of its elements would qualify for this label, but such a distinction would add little to the objective of providing a probabilistic framework for the Semantic Web.

In other words, merely adding probabilities to concepts does not guarantee interoperability with other ontologies that also carry probabilities. Clearly, more is needed to justify a new category of ontologies, and such extra justification doesn't come from the syntax used for including probabilities.

***Definition 3:*** A *probabilistic ontology* is an explicit, formal knowledge representation that expresses knowledge about a domain of application.

This includes:

- 2.a) Types of entities that exist in the domain;
- 2.b) Properties of those entities;
- 2.c) Relationships among entities;
- 2.d) Processes and events that happen with those entities;
- 2.e) Statistical regularities that characterize the domain;
- 2.f) Inconclusive, ambiguous, incomplete, unreliable, and dissonant knowledge related to entities of the domain;
- 2.g) 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 embedded to that knowledge in a principled, structured and sharable way, ideally in a format that can be read and processed by a computer. They also expand the possibilities of standard ontologies by introducing the requirement of a proper representation of the statistical regularities and the uncertain evidence about entities in a domain of application. Yet, meeting the main objective of this research effort requires going a step further and also allowing for reasoning upon what now can be represented via probabilistic ontologies.

In the current SW's scheme, OWL ontologies are used for representing domain information in a way to enable Web services/agents to perform logical reasoning over that information. More specifically, ontologies intended to facilitate logical reasoning by

Web services/agents are commonly written using OWL-DL, the decidable subset of OWL language that is based on Description Logics. Writing ontologies in OWL-DL permits the use of DL reasoners such as Racer (Haarslev & Möller, 2001) to perform logical reasoning over its contents. Figure 19 depicts the typical flow of knowledge of a Web agent that is based on logical reasoning.

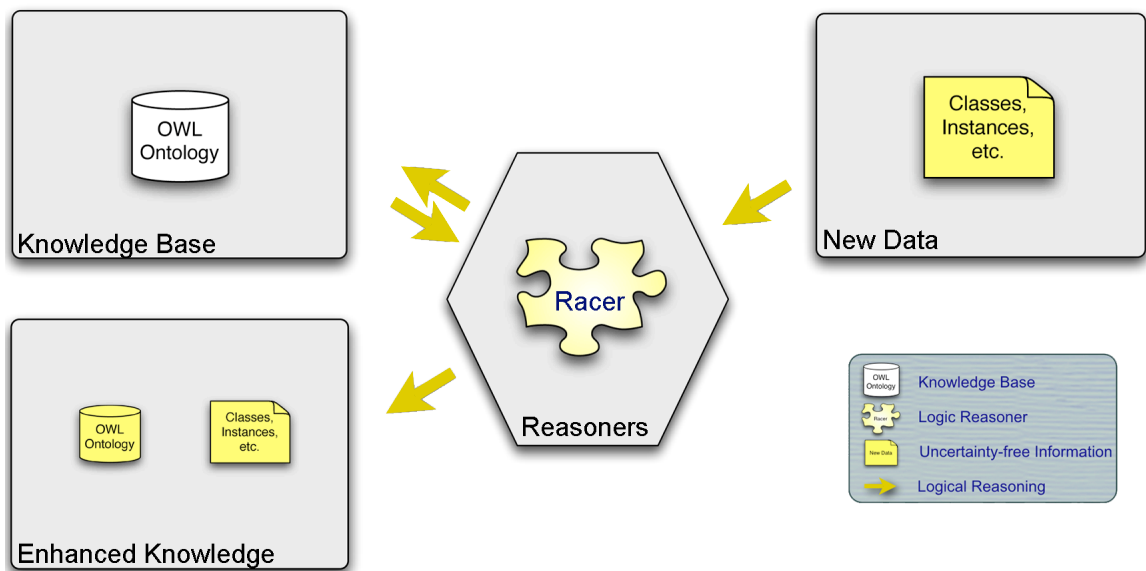


Figure 19. Typical Web Agent's Knowledge Flow – Ignoring Uncertainty

In the situation depicted by the figure, the Web agent (here assumed as using Racer as its reasoning engine) receives new data and uses the domain information stored in the knowledge base (an OWL ontology) to perform logical reasoning. Its output is the enhanced knowledge that results from the logical reasoning process, and can be used as a means to answer queries posed to the Web agent and/or to update the knowledge base. As an example from our case study, suppose that a logical reasoner receives information that a newly detected starship is a (say) Keldon-Class Warship operated by the Cardassian species. Then, it verifies the axioms and restrictions conveyed in the Enterprise's Star

Trek ontology that an individual of class *Starship* possessing these properties could only be a foe. As a result, the reasoner classifies that individual as being a member of the subclass *Foe* and returns the new knowledge to the system.

Among the possibilities of logical reasoning is the ability to infer whether a given concept is a subclass of another (i.e. subumption), whether the definitions of a class will make it impossible to have any instances (i.e. consistency), and others that make OWL-DL ontologies very a powerful tool for the SW. Not surprisingly, most SW ontologies are being developed using OWL-DL.

Still, as we have emphasized in the previous Chapters, the above-mentioned features of logical reasoning are only possible when complete information is available to the reasoner. In our example, factors such as distance, ambiguities on the received data, overlapping characteristics among starship classes and other sources of uncertainty would most likely prevent any definitive conclusion about the starship type or operator species to be drawn.

Sources of uncertainty are the rule in open environments such as the SW, which reinforces the use of probabilistic ontologies for both representing uncertain knowledge and reasoning with it. Figure 20 depicts the same Web agent's knowledge flow, but this time incorporating the concepts of a probabilistic ontology and a plausible reasoner.

As it is shown in the picture, the knowledge base now consists of a PR-OWL ontology. This expanded depiction of the starship domain includes all the concepts of the previously depicted OWL ontology plus the uncertain information that could not be expressed in a principled way with a standard OWL ontology. Also, new evidence that

would be simply discarded by the logical system (e.g. the recently detected starship has 90% chances of being operated by Cardassians) can now be accepted and considered in the reasoning process. Since the Web agent now uses a probabilistic reasoner (Quiddity\*Suite in this example), each and every piece of evidence would be used to upgrade the system's knowledge. In short, all the advantages of a Bayesian probabilistic system that were covered in the previous chapters are now available.

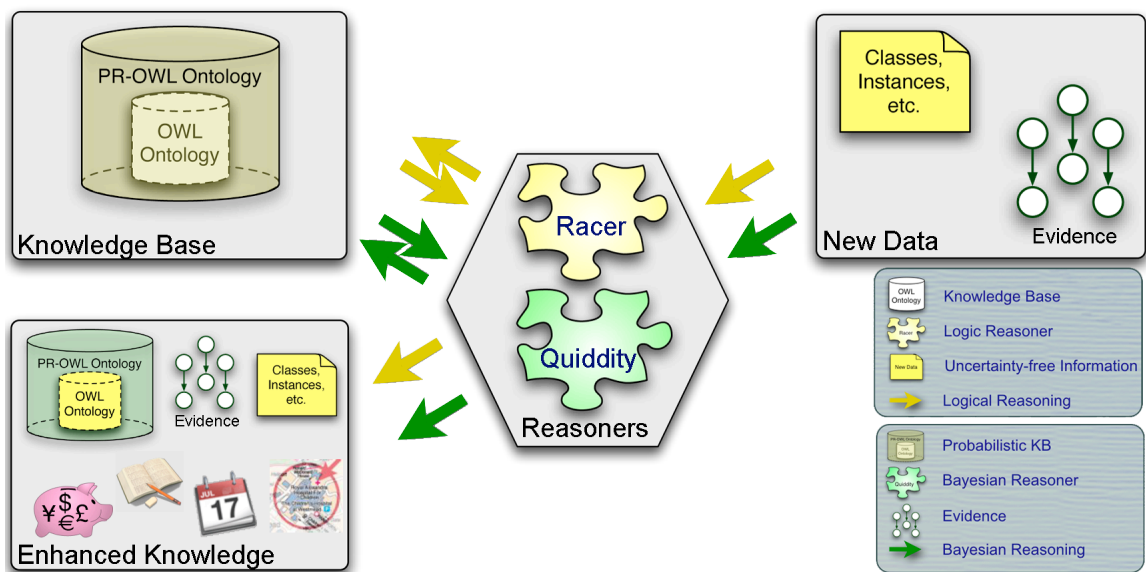


Figure 20. Typical Web Agent's Knowledge Flow – Computing Uncertainty

The result of the added capabilities, as implied in the enhanced knowledge box, is the system's ability to keep the best estimate possible for its queries given the previous knowledge and all the available data at any given time. This new aspect opens the opportunity for solving many SW problems that logical systems so far have been unable to solve, such as schedule matching, optimal decision with incomplete data, etc.

In the scheme depicted in Figure 20, there are two aspects that cannot be implemented today. First, there is no Semantic Web language capable of representing

probabilistic ontologies in a principled way and, second, after such a language is developed there will be no probabilistic reasoner specifically designed to perform the reasoning services using the newly developed language.

In this dissertation, we have tackled the first aspect with the development of PR-OWL, while addressing the second problem by defining a set of rules for translating a MENB model to a Quiddity\*Suite model. This was not a trivial process, and the following sections are intended to explain what we have done to “clear the path” for probabilistic ontologies.

Our first major issue was the fact that standard MEBN does not have built-in support for the complex elements of the OWL language, preventing any attempts of a direct PR-OWL implementation. For instance, standard MEBN is untyped, which means there is no built-in support for PR-OWL probability distributions to depend on types, or for representing type uncertainty, both highly desirable features for a probabilistic ontology. In section 4.1 we present our solution to this problem: the development of an extended version of MEBN logic that incorporates all the desirable features missing in the standard logic.

The second and last Section of this chapter is devoted to explain how we addressed another major issue preventing us to develop PR-OWL: there is no “off-the-shelf” probabilistic reasoner that implements all MEBN features. By the time of our research, Quiddity\*Suite was rapidly reaching a stage in which most if not all MEBN features would be implemented, so we developed a set of rules that allowed us to use it as a MEBN implementation and a valid PR-OWL reasoner.

#### 4.1 A Polymorphic Extension to MEBN

The most obvious difference between a typed and an untyped logic is the addition of a type label declaration. However, just adding a type label to MEBN logic will not provide it with the advantages of type systems, such as efficient inference based on inheritance, increased readability, conciseness, etc. Developing a typed version of a probabilistic first-order logic involves creating a coherent set of standard definitions and inference rules that collectively form a consistent type system.

A common way to declare types is to define a monadic predicate for each type; the predicate is true of entities of the given type and false otherwise. According to Sowa (2000, page 473), a type system adds no expressive power to a knowledge representation language, in that every theorem and proof in a typed logic will have a counterpart in its untyped version. Given the formal equivalence in expressiveness between typed and untyped logics, one might wonder why typed logics are so much more popular. This popularity can be explained by their advantages in terms of tractability and ease of use.

FOL provides the theoretical foundation for the type systems used in popular object-oriented and relational languages. The popularity of typed logics reflects the prevalence of types in informal reasoning. Classification of objects in terms of the purposes for which they are used typically results in a more or less well-defined type system (Cardelli & Wegner, 1985). The advantages of languages based on a typed logical system are usually related to code optimization and to less error-prone syntax. A number of authors have developed semantics for typed languages (Milner, 1978; Damas & Milner, 1982; Mitchell, 1984).



Standard MEBN is untyped, but a typed extension can provide a sound mathematical basis for representing and reasoning under uncertainty in typed domains, including domains of unbounded or infinite cardinality. Among the advantages of a MEBN-based type system is the ability to represent *type uncertainty*. As an example, suppose there are two different types of space traveling entities, starships and comets, and incomplete information is given about the type of a given entity. In this case, the result of a query that depends on the entity type will be a weighted average of the result given that the entity is a comet and the result given that it is a starship.

Further advantages of a MEBN-based type system include the ability to refine type-specific probability distributions using Bayesian learning, assign probabilities to possible values of unknown attributes, reason coherently at multiple levels of resolution, and other features related to representing and reasoning with incomplete and/or uncertain information.

Therefore, defining standard syntax and semantics for a MEBN-based type system would combine the advantages of FOL-based type systems with the ability to express and reason with uncertainty, including uncertainty about the type of an entity.

In order to provide a typed version of MEBN, two main changes are proposed here: (1) modify the definition of an MTheory provided in Laskey (2005) to allow for a random variable to have multiple home MFrag (i.e. polymorphism), and (2) include a set of built-in MFrag that provide a standard procedure for defining domain-specific types and subtypes.

As a basic assumption, types are arranged in a tree-structured hierarchy (thus excluding multiple inheritance for the present). The types immediately below a given type (e.g. *Starship*) are called its *subtypes* (e.g. *MilitaryStarship* and *CivilianStarship* are subtypes of *Machine*). The next-higher type to a given type is called its *parent type* (e.g. the parent type of *MilitaryStarship* is *Starship*). For our present proposed type system, a given type can have only one parent type. It is relatively straightforward to extend the system being presented here to a type system with multiple-inheritance.

Figure 21 shows the *Star Trek* MTheory from Figure 12 with the addition of the Transporter Mfrag, illustrating the effects of a polymorphic, typed version. This new generative Mfrag conveys information on *Enterprise's* ability to beam a person or an object to a close planet.

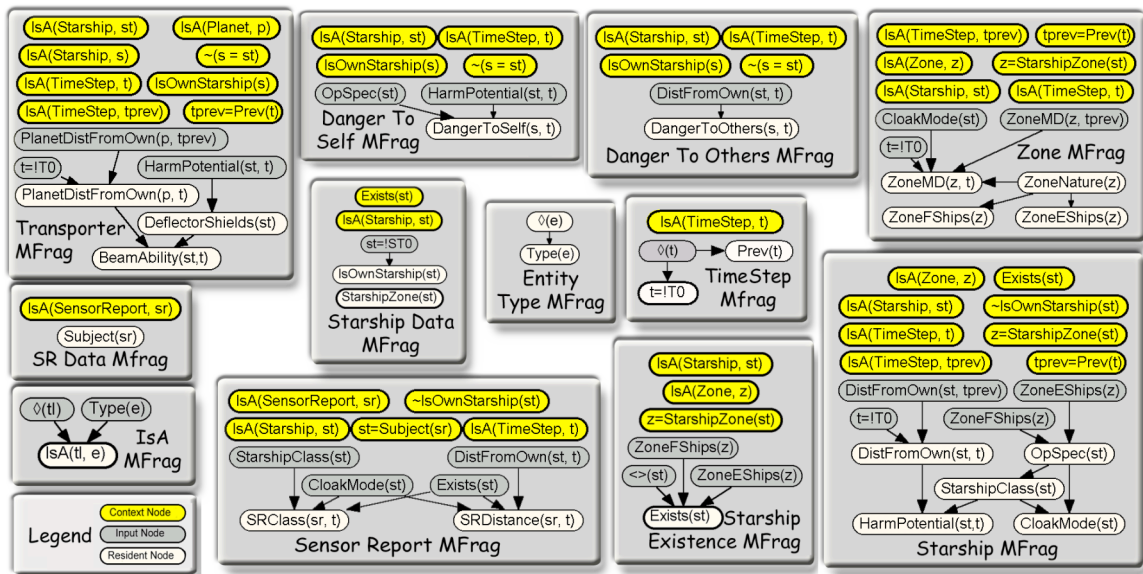


Figure 21. Star Trek MTheory with the Transporter Mfrag – Untyped Version

A Transporter is the device that performs the beaming process (in “theory, a form of molecular transport), which depends on the distance between *Enterprise* and the target

planet, and on whether the defense shields are activated. Usually, the shields will be up when the potential harm of any starship on the vicinities of *Enterprise* reaches a certain threshold (inferred by the instances of node *HarmPotential*(*st*, *t*) for each nearby starship).

Both MTheories have an Entity MFrag used for declaring the possible types of entity that can be found in the model, and an IsA MFrag for allocating a type label to a given entity. This is an example of a situation in which the modeler wanted a type system so she had to build her own scheme for defining entity types. Given the popularity of type system, we might expect other modelers to specify their own Type and IsA MFrag, possibly resulting in many different, incompatible implementations of MEBN-based type system. A standard typed extension to MEBN frees each implementer from having to define her/his own type systems and thus supports a greater level of interoperability.

Even with the modeler defining her/his own type system, situations in which a node such as *PlanetDistanceFromOwn*(*p*, *t*) cannot be named *DistanceFromOwn*(*p*, *t*) because there is another node with that name are very likely to happen. According to the unique home MFrag restriction, one node cannot have two home MFrag so a different name would have to be issued for any node that measures the distance from an object to the *Enterprise*. In short, the MTheory in Figure 21 does not have subtyping, polymorphism, or inheritance, features that are often useful for modeling complex, real-life problems. As shown in the next section, the extended version proposed in this work includes these features.

#### 4.1.1 The Modified MTheory Definition

In order to provide a typed version of MEBN logic, the definition of a generative MTheory is modified here to allow a random variable to have more than one home MFrag. The unique home MFrag restriction is relaxed to allow multiple home MFrag for a node, provided that if a node is resident in two MFrag then either: (1) the two contexts in the different MFrag are entirely disjoint, or (2) one context is strictly contained in the other. Distributions defined for a type are inherited for all its subtypes, except that distributions defined in more general contexts are overridden by distributions defined in more restricted context.

As a means of providing a standard support for typing, typed MEBN includes the four MFrag depicted in Figure 22 among the built-in MFrag.

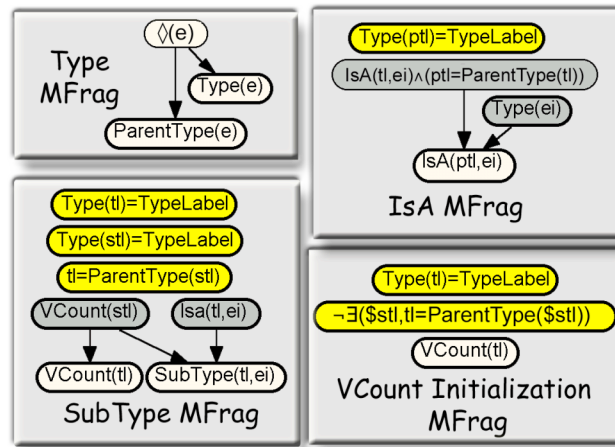


Figure 22. Built-in MFrag for Typed MEBN

**The Type MFrag.** The *Type* MFrag has three resident nodes and no input or context nodes. It lays out the core structure for the type system and provides the basic support for the domain-specific type definitions. The uppermost resident node,  $\Diamond(e)$  (the identity random variable), is parent of the other two nodes and has all valid entities of a

given domain as its states. As with standard MEBN, the only values that may have non-zero probability are  $e$  (for meaningful entities) and  $\perp$  (for identifiers that refer to meaningless or nonexistent entities).  $Type(e)$  is domain-specific (i.e., defined by the knowledge engineer) and its states include three special types that are standard to all polymorphic MEBN models plus the domain-specific types defined by the knowledge engineer.

The special types are: (1) *TypeLabel*, which includes the labels of all domain-specific types and subtypes; (2) *Boolean*, which includes the truth-values T, F, and  $\perp$ ; (3) *CategoryLabel*, which includes labels for RVs whose domain is a list of categorical values; and (4) *PositiveNumber*, which is used for virtual counts (see below).

Finally,  $ParentType(e)$  defines the type of which  $e$  is a subtype. When  $e$  is replaced by a unique identifier of a *TypeLabel* entity, then  $ParentType(e)$  must have a unique *TypeLabel* entity as its value if the substituting entity type has a parent type. In addition, if  $e$  is replaced by a unique identifier of a *TypeLabel* entity at the level of the type hierarchy immediately below the root *Entity*, then  $ParentType(e)$  will have value *Entity*; and furthermore,  $ParentType(Entity) = Entity$ .

**The IsA MFrag.** The *IsA* MFrag is the home MFrag for  $IsA(ptl, ei)$ , where  $ei$  is an instance of type  $ptl$ . As an example, suppose we replace  $ptl$  with the unique identifier corresponding to the *Starship* type label. If we then replace  $ei$  with the unique identifier of a starship entity (say !ST1), then  $IsA(ptl, ei)$  becomes  $Isa(Starhship, !ST1)$ , which has value T. Conversely, if the variable  $ei$  is replaced with a unique identifier of a planet entity (say !P1) then  $IsA(ptl, ei)$  becomes  $Isa(Planet, !ST1)$ , which has value F.

The *IsA* MFrag has only one context node, which is satisfied when the variable *ptl* is replaced by a *TypeLabel* entity. The MFrag's only resident node,  $IsA(ptl, ei)$ , is true when *ei* represents an instance of the type represented by *ptl*. This random variable has two parents, the input nodes  $Type(ei)$  (the type of *ei*) and the Boolean RV term  $Isa(tl, ei) \wedge (ptl = ParentType(tl))$  (*ei* represents an instance of *tl* and *tl* is a subtype of *ptl*). Its value is T when either of its parents is true. That is,  $IsA(ptl, ei)$  is true when  $Type(ei)$  is *ptl* or (recursively) when  $Isa(tl, ei)$  is true for a type *tl* that is a descendant of *ptl* in the type hierarchy. The distribution for the Boolean RV term  $Isa(tl, ei) \wedge (ptl = ParentType(tl))$  is defined via the built-in logical MFrag.

We noted above that type systems are typically defined by specifying a unary predicate for each type. Our *IsA* predicate is binary. We have taken advantage of polymorphism to define a single binary predicate  $Isa(tl, ei)$  rather than a unary predicate for each type (e.g.,  $IsaStarship(ei)$ ,  $IsaPlanet(ei)$ , etc.)<sup>24</sup>

**The SubType MFrag.** The three upper, unconnected nodes in the *SubType* MFrag of Figure 22 are context nodes specifying that the variables *tl* and *stl* are placeholders for unique identifiers representing *TypeLabel* entities and that the first *TypeLabel* (referred to by *tl*) is the parent type of the second (referred to by *stl*). The rightmost input node,  $IsA(tl, ei)$ , has its local distribution defined in the *IsA* MFrag above, so its value will be T when the unique identifier replacing *ei* refers to an entity whose type is either *tl* or one of its descendant types. The other input node,  $VCount(stl)$ , has a

---

<sup>24</sup> Even though the MTheory in Figure 12 was built using the standard version of MEBN, for simplicity and in order to facilitate its translation to Quiddity\*Suite we implicitly allowed polymorphism in the specific case of the *IsA* MFrag (i.e. by making them binary predicates in that model).

positive number as its value, and represents the relative probability that an entity of type  $ParentType(stl)$  is an instance of  $stl$ . That is, if our only information about an instance  $ei$  is that its type is a subtype of  $tl$ , and if  $stl1$  and  $stl2$  are labels for subtypes of  $tl$ , the ratio of the probability of  $Isa(stl1, ei)$  to the probability of  $Isa(stl2, ei)$  is given by  $VCount(stl1)/VCount(stl2)$ .

The name  $VCount$  stands for “virtual count,” a term used in the literature on Bayesian learning to refer to parameters in a prior distribution that correspond roughly to remembered prior observations. That is, we can think of  $VCount(tl)$  as the number of instances of  $tl$  that have been encountered in previously experienced situations (although there is no requirement that its value be an integer). Virtual counts are important for representing type uncertainty.

Because virtual counts behave like remembered counts, the virtual count for a type is constrained to be equal to the sum of the virtual counts for its subtypes. The distribution for the resident node  $VCount(tl)$  enforces this constraint. This MFrag defines virtual counts only for non-leaf nodes in the type hierarchy. Virtual counts for leaf nodes are defined in the virtual count initialization MFrag described below. The context constraint  $tl=ParentType(stl)$  ensures that  $tl$  is not a leaf node in the type hierarchy. The distribution of  $VCount(tl)$  places probability 1 on a value equal to the sum of the virtual counts for subtypes of  $tl$ . Although virtual counts are not required to be integers, as for all MEBN random variables, the possible values must be a countable set.

The resident node  $SubType(tl, ei)$  has as its possible values all entities of type  $TypeLabel$  other than the type label  $Entity$ , along with the value  $\perp$  (absurd). When the

parent node  $IsA(tl, ei)$  has value  $F$ , then  $SubType(tl, ei)$  has value  $\perp$  (i.e., whether an entity is an instance of something that is not a type is an absurd question). If  $IsA(tl, ei)$  has value  $F$ , then the distribution for  $SubType(tl, ei)$  puts non-zero probability only on values for which there is a  $VCount(stl)$  parent (these are  $stl$  for which the context constraint  $tl=ParentType(stl)$  is met, i.e., subtypes of  $tl$ ). The probabilities of the subtypes are proportional to their virtual counts.

**The VCount Initialization MFrag.** The context nodes for this MFrag ensure that the variable  $tl$  is replaced by the unique identifier of an entity that (1) is of type *TypeLabel* and (2) has no subtypes (i.e. it is a leaf node in the type hierarchy). The context node  $\neg\exists(\$stl, tl=ParentType(\$stl))$  represents the negation of an existentially quantified first-order sentence. It is satisfied when there is no entity instance of which  $tl$  is a parent type. The symbol  $\$stl$  is a *Skolem term*, which represents a generic instance that satisfies the sentence if it is satisfiable and otherwise has value  $\perp$  (Laskey, 2005). We note that one of the conditions of the theorem of Laskey (2005) that a conditional distribution exists on interpretations of any consistent, finitely axiomatizable first-order theory is that no context RV may contain quantifier random variables. If there are only finitely many types, quantified statements about type labels can be treated as shorthand notation for finite conjunctions and disjunctions. Thus, the theorem still holds for typed MEBN.

The distribution of the resident node  $VCount(tl)$  specifies a probability distribution for the leaf node  $tl$  in the type hierarchy. The distribution of  $VCount(tl)$  for leaf nodes  $tl$  in the type hierarchy is supplied by the domain expert.



Note that the random variable  $VCount(tl)$  has two resident MFrag, thus violating the original conditions for a valid MTheory (Laskey 2005). However, it satisfies the conditions for extended version defined above, because the contexts for the two home MFrag are disjoint.

#### 4.1.2 The Star Trek MTheory Revisited

The MTheory of Figure 23 has takes advantage of our extended version of MEBN, but is equivalent, *mutatis mutandis*, to the MTheory of Figure 21. The most obvious modification is the absence of the Type and the Isa MFrag. Also, the new version allows several similar random variable labels to be combined into a single label, such as in the case of nodes *DistFromOwn()* in the Transporter and the Starship MFrags.

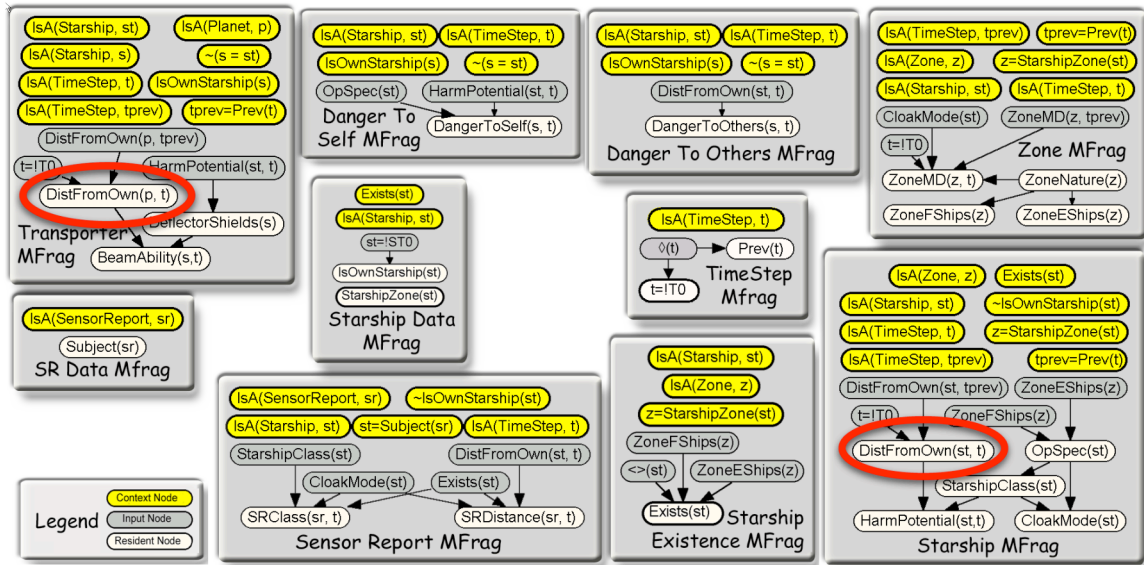


Figure 23. Star Trek MTheory with the Transporter MFrags – Typed Version

It would be straightforward to add Starship subtypes (e.g., *MilitaryStarship* and *CivilianStarship*) to this MTheory. Distributions defined for entities of type *Starship*

would be inherited by entities of type *MilitaryStarship* and *CivilianStarship* unless overridden.

The ability to use the same name for similar concepts applied to different types of entities is very useful for the knowledge base designer. It supports portability and reusability, allows for more natural naming conventions, supports more compact representations, and helps to prevent errors. Less obvious potential gains include savings in memory allocation, the possibility of optimizing compilers and reasoners to exploit the type structure, and other advantages of standard type systems. These advantages can now be applied to a probabilistic first-order logic.

#### 4.2 Using Quiddity\*Suite for Building SSBNs

Information Extraction and Transport's (IET) Quiddity\*Suite™ is a probabilistic frame-based modeling toolkit that implements many features of MEBN logic and supports type uncertainty and multiple inheritance. Quiddity\*Suite has been applied to a wide range of problems ranging from visual target recognition to multi-sensor data fusion to dynamic decision systems in the C3I arena (Fung *et al.*, 2004).

A *frame* is a knowledge representation structure that expresses a concept or a situation (Minsky, 1975), and it was a rather novel approach to knowledge representation for a period in which rule-based or logic-based were the predominant formalisms. Frame-based systems allow knowledge builders to easily describe the types of objects in a domain. As such, they provide the conceptual basis for expressing knowledge in an object-oriented way that inspired many subsequent formalisms (e.g., Bobrow &

Winograd, 1977; Brachman & Schmolze, 1985; Kifer *et al.*, 1990; Greco *et al.*, 1992). According to Fikes & Kehler (1985), frames represent entities or classes of entities, and can incorporate sets of attribute descriptions called *slots*. Also, slots can have a set of properties, which are called *facets* of a slot.

In standard frame languages, there is no pre-defined way to express uncertainty about values of an attribute. Unless a slot is left unassigned, either the value must be known or a default value should be used. Quiddity\*Suite expresses uncertainty about the value of an attribute by associating a random variable with each slot for which the value may be uncertain. When an instance of the frame is created, a random variable is created for each uncertain slot. In order to carry the information needed to define probability distributions for the random variables associated with uncertain slots, Quiddity\*Suite uses a set of pre-defined facets, which represent possible states, parents, and conditional distribution given parents.

A detailed coverage on Quiddity\*Suite's approach for representing uncertainty in frame-based systems is outside the scope of this work, but in our research experiments we were able to use it as a means to perform SSBN construction from evidence applied to a generative MTheory.

As a proof of concept on the feasibility of using Quiddity\*Suite as a partial implementation of MEBN, we have translated the Star Trek MTheory into Quiddity\*Suite format. The resulting model is capable of building any SSBN that can be generated from the original Star Trek MTheory. The source code for the model is presented in Appendix A. In this Section; we describe the translation process

emphasizing which features could be directly translated and which required some additional effort. Also, we address some of the issues and details that should be taken into account when translating MTheories to Quiddity Models.

Readers should have in mind that the next Subsections are not meant to be a Quiddity\*Suite tutorial or learning asset. Instead, the main purpose is to show how the concepts in MEBN logic translate (or not) to Quiddity\*Suite elements. Even though knowledge on Quiddity\*Suite is necessary for understanding/applying the general translation rules presented below, readers that are familiar with frame systems notation would find relatively easy to understand the general idea of the translation procedures.

#### 4.2.1 Concepts with Direct Translation

In the typed version of MEBN logic developed for this research, every domain-specific entity has a type, which is represented as one of the possible values of node *Type(e)* in the built-in Type MFrag). By definition, all possible values of node *Type(e)* have their type defined as *TypeLabel*, which is itself a built-in possible value or *Type(e)*. As an example, the Starship MTheory has four types of entities (starships, sensor reports, zones, and time steps), so the possible states of its built-in node *Type(e)* are the special built-in states plus the domain-specific states *Starship*, *SensorReport*, *Zone*, and *TimeStep*. Also, the typed version of MEBN logic presented here also supports subtyping, so we could easily create a type hierarchy in which starships would have subtypes (say) *military* and *civilian*.

Quiddity\*Suite represents entity types as frames, and also supports frame subtyping. An entity type in MEBN logic corresponds to a frame in Quiddity\*Suite. Thus, a

good way of enforcing compatibility between models based on MEBN logic and their counterparts in Quiddity\*Suite is to use an approach that makes such correspondence more explicit.

As explained in Chapter 3, MEBN logic allows multiple, equivalent ways of portraying the same knowledge (recall example in Figure 13). Therefore, MEBN modelers willing to achieve full compatibility with Quiddity\*Suite (and with frame systems in general) are encouraged to use the object oriented approach we sought with the Star Trek MTheory, which used the concept of an *entity cluster*.

**Definition 3:** An *entity cluster* is a group of MFrag within a generative

MTheory having the following characteristics:

- 3.a) In any MFrag contained in the entity cluster, there is an ordinary variable, called the *subject argument* of the MFrag, such that any non-constant random variable in the MFrag has the subject argument as one of its arguments.
- 3.b) The context constraints of each MFrag in the entity cluster specify the type of the subject argument. This type is called the *subject type* of the MFrag.
- 3.c) The subject types of all MFrag in the entity cluster are the same. ■

The above definition addresses only the top-level classes. That is, if entity clustering is desired for subclasses then this definition will have to be extended to accommodate subtyping. As an example, if we had different subtypes of starship, we

might have entity clusters that contained definitions for some nodes at the supertype level, and other nodes at the subtype level. However, formalizing the translation rules for subtyping is a subject for future work.

Figure 24 depicts the Star Trek MTheory divided by its entity clusters. Building an MEBN model using the entity clusters approach facilitates the interoperability among different modeling tools. In the specific case of the Star Trek MTheory, it allows a direct mapping between frames and domain-specific entities (which have all of its attributes within the same entity cluster). In addition, using this modeling approach makes it easier to keep MEBN logic's flexibility to display the same information in different MFrag configurations. As an example, depending on the model objectives, the Starship MFrag in Figure 24 could be easily replaced with the three equivalent MFrams in Figure 13.

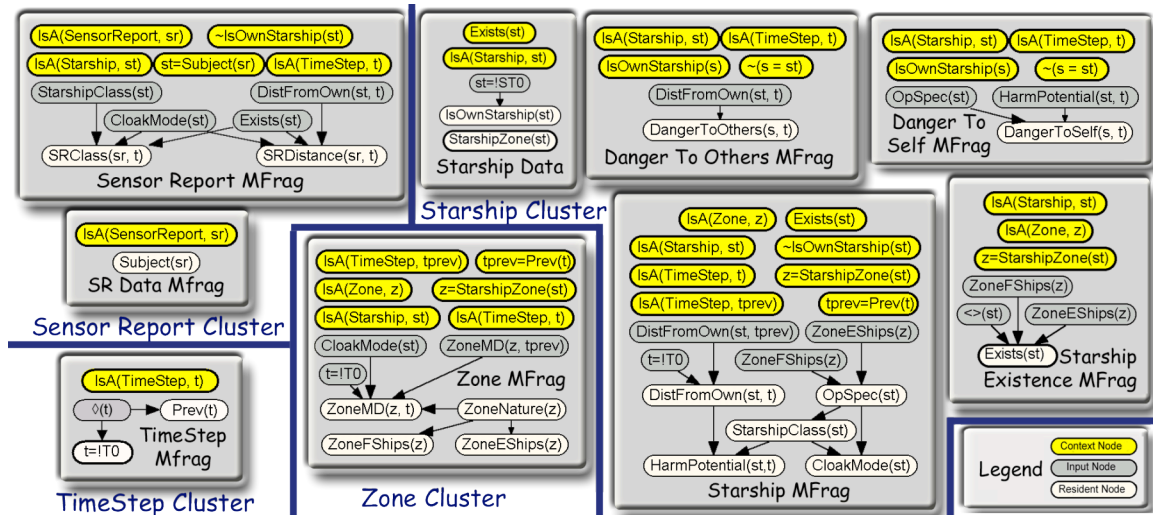


Figure 24. Entity Clusters of Star Trek MTheory

Using the entity cluster modeling approach, most concepts in MEBN logic can be easily translated to Quiddity\*Suite syntax. Within a given entity cluster, all resident nodes are directly mapped as slots of the frame that corresponds to that entity cluster.

Input nodes are mapped in accordance with the MFrag in which they are resident nodes. If an input node is a “copy” of a resident node defined in an MFrag within the entity cluster, then it is listed directly in the Parents facet of their children. Input nodes defined outside the entity cluster are also listed in the Parents facet of their children, but have their name preceded by a pointer slot. A pointer slot is a slot that has another frame as its domain, and it is used for making references to that frame.

Figure 25 exemplifies the above explanation using the Sensor Report entity cluster depicted in the MTheory in Figure 24. The cluster’s name is directly used in the frame (bullet 1 in the figure), and each of the resident nodes of the cluster’s two MFrag is transformed into a slot in that frame (bullets 2 to 4). It is important to note that node *Subject(sr)* is an attribute of a sensor report that links each instance of a sensor report to its respective subject. In our model, subjects of sensor reports are starships and thus the possible states of *Subject(st)* are starship entities. Therefore, slot subject has frame Starship as its domain and works as a pointer to that domain.

The use of a pointer is easily observable in this example, since all the input nodes were defined outside the Sensor Report entity cluster. That is, there are no parent resident nodes and also no input nodes defined within the cluster. In this case, each input node is defined in the Parents facet of its respective children with the pointer node’s name preceding it (see bullet 5 for the parents of SRCClass).

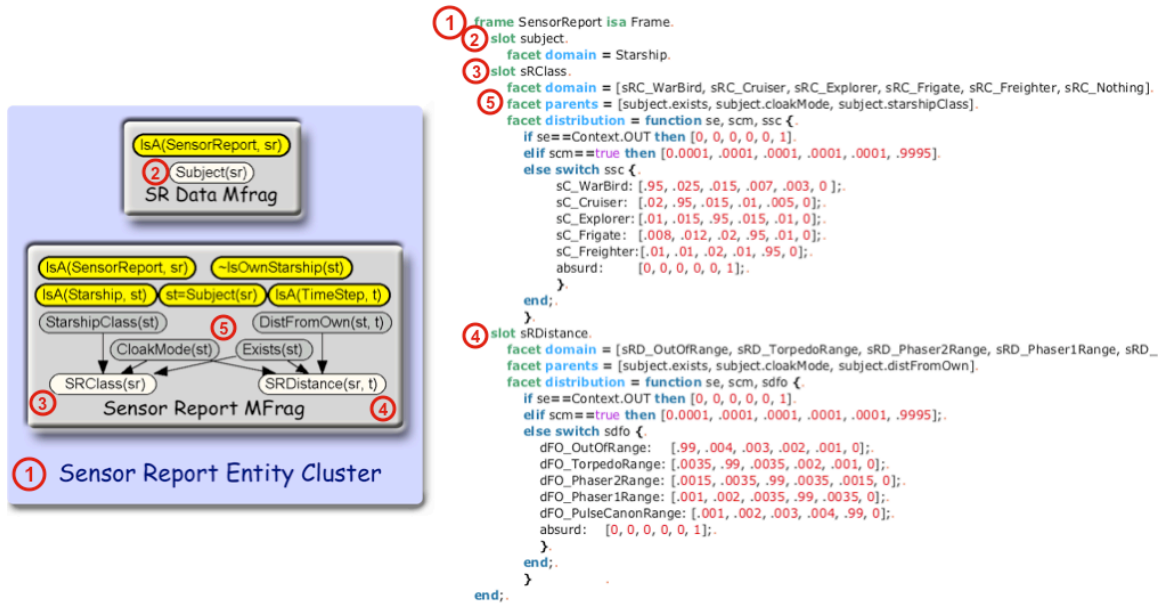


Figure 25. Mapping the Sensor Report Entity Cluster to a Frame

Although not emphasized in Figure 25 for the sake of simplicity, the probability distribution and the states of a resident node are directly translated to its respective slot's distribution and domain facets respectively. Quiddity\*Suite's syntax provide a rich list of possibilities for portraying a probability distribution via the distribution facet, from built-in standard distributions (e.g. UniformDiscreteDistribution for the discrete uniform distribution) to highly complex combinations of functions. Also, there are different possibilities for defining the domain of a slot via the domain facet, but covering all the possibilities is outside the scope of this dissertation. Table 3 summarizes the concepts discussed above, which can be directly translated from MEBN to Quiddity\*Suite, and presents some examples.



Table 3. MEBN Elements Directly Translated into Quiddity\*Suite

MEBN Concept	Quiddity*Suite Representation	Quiddity*Suite Examples (from the Star Trek MTheory)
Entity Type	Frame	<i>frame</i> Starship <i>isa</i> Frame
Type hierarchy	Frame hierarchy.	<i>frame</i> MilitaryStarship <i>isa</i> Starship
Resident nodes	Slots	<i>slot</i> z_ZoneEShips
Parent resident and input nodes defined within the entity Cluster	Parents facet	<i>facet parents</i> = [z_ZoneNature]
Parent input nodes defined outside the entity cluster	Parents facet + pointer Slot	<i>facet parents</i> = [PointerSlot.z_ZoneNature]
Probability Distribution	Distribution facet	<i>facet distribution</i> = <Quiddity table or formula> <i>facet distribution</i> = MaxDistribution
States	Domain facet	<i>facet domain</i> = booleandomain <i>facet domain</i> = Starship <i>facet domain</i> = [ <list of states> ]

#### 4.2.2 Concepts with a More Complex Translation

In Subsection 4.2.1 we intentionally omitted some MEBN concepts that do appear in the Star Trek MTheory, most notably the context nodes and the ordinary variables. Also, we avoided specific cases of the concepts already cited, such as situations in which an input node might generate multiple instances of itself. The reason of those omissions is the intrinsic complexity of those cases, which demands a more elaborate discussion.

Context nodes are a powerful aspect of MEBN logic that allow specifying carefully defined situations in which a given MFrag is valid. Some of the more common

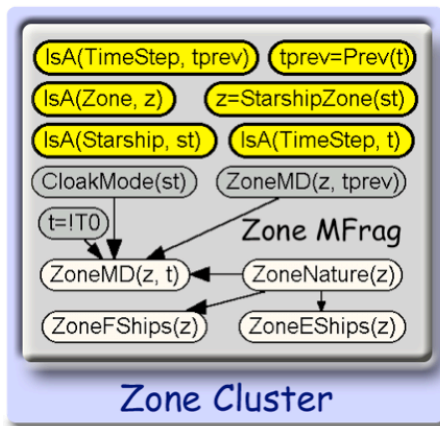
context nodes are relatively easy to express in Quiddity\*Suite, while others present some issues requiring relatively complex workarounds. Those cases are addressed in this Subsection.

The most common use of context nodes involves the specification of the types of the entities in a given MFrag. In an entity cluster structure, all MFrams have the cluster's subject entity as their main attribute, so they will include an ordinary variable representing that subject entity and a context node making that representation explicit. A brief verification on the Star Trek MTheory in Figure 24 will show that all MFrams within each cluster have a node such as *IsA(SubjectEntity, ordvar)*. As an example, all MFrams within the Starship cluster have an *IsA(Starship, ...)* context node.

The other *Isa(TypeLabel, ordvar)* in each MFrag are used to define the type of the instances that can substitute each ordinary variable. As a practical rule, for each of those "extra" *Isa(TypeLabel, ordvar)* a pointer slot would have to be created in the frame. Of course, only one pointer slot is needed so if there is one already then it is not necessary to create another. As an example from the Sensor Report cluster, since there is already a slot pointing to frame Starship (directly created because of the *Subject(sr)* resident node) then when evaluating the context node *IsA(Starship, st)* in the Sensor Report MFrag there will be no need to create another pointer slot.

One exception to the above general rule is the *IsA(TimeStep, ordvar)* context nodes, which should not generate a pointer slot. The intended meaning of an entity of type TimeStep is to indicate time recursion, which has a special treatment in Quiddity\*Suite. Figure 26 illustrates an example of an MFrag (Zone, the unique MFrag

in Zone Cluster) with temporal recursion and its respective frame counterpart. There are three context nodes and two input nodes expressing that recursion in MEBN. The two *IsA(TimeStep, ordvar)* context nodes are used to define the type of the ordinary variables *t* and *tprev*, while the remaining context node uses the *Prev(t)* random variable to specify that variable *tprev* is the predecessor of variable *t* in the ancestor chain of that time recursion. The two input nodes are *t = !T0* and *ZoneMD(z, tprev)*. The first “anchors” the recursion, while the latter makes it explicit that the distribution of *ZoneMD(z, t)* depends on its immediate ancestor in the recursion *ZoneMD(z, tprev)*. These five nodes are expressed by two specific elements in the ZoneMD slot. The first is its Parents facet, which contains zoneMD.PREV, where the suffix .PREV indicates that this slot’s distribution depends upon its ancestor. The second is the initialState facet, which contains the distribution for the first instance of the ancestor chain.



```

frame Zone isa Frame.
slot starship.
  facet domain = Starship.
  facet distribution = UniformDiscreteDistribution.
slot anyStInCloakMode.
  facet domain = [false, true].
  facet parents = [starship.cloakMode].
  facet distribution = MaxDistribution.
slot zoneNature.
  facet domain = [zN_BlackHoleBoundary, zN_DeepSpace, zN_PlanetarySystem].
  facet distribution = [.05, .80, .15].
slot zoneEShips.
  facet domain = [zES_0, zES_1, zES_2, zES_3, zES_MoreThan3].
  facet parents = [zoneNature].
  facet distribution = function zn {
    zN_BlackHoleBoundary: [.95, .03, .01, .007, .003];
    zN_DeepSpace: [.98, .01, .007, .002, .001];
    zN_PlanetarySystem: [.60, .15, .12, .08, .05];
  }
slot zoneFShips.
  facet domain = [zFS_0, zFS_1, zFS_2, zFS_3, zFS_MoreThan3].
  facet parents = [zoneNature].
  facet distribution = function zn {
    zN_BlackHoleBoundary: [.99, .005, .0035, .001, .0005];
    zN_DeepSpace: [.98, .01, .007, .002, .001];
    zN_PlanetarySystem: [.50, .20, .15, .10, .05];
  }
slot zoneMD.
  facet domain = [zMD_Low, zMD_Medium, zMD_High].
  facet parents = [zoneNature, anyStInCloakMode, zoneMD.PREV].
  facet initialState = [.70, .20, .10].
  facet distribution = <OMMITTED FOR BREVITY>.
end;

```

Figure 26. Zone Entity Cluster

As a general rule, MEBN recursions will follow this pattern of three context nodes defining two ordinary variables and their precedence, and two input nodes for “anchoring” the recursion and declaring the recursive resident node’s dependence over its predecessor. Such pattern should then be translated to Quiddity\*Suite using the .PREV suffix and the initialState facet accordingly.

Apart from defining types of ordinary variables and establishing recursive patterns, context nodes are used to specify how the process of SSBN construction will occur. In other words, context nodes can be seen as logic rules that define the conditions under which instances of the random variables of an MFrag will be created during the process of SSBN construction. As an example from the Zone MFrag of Figure 26, the context node  $z=StarshipZone(st)$  specifies a restriction on instances of Zone and Starship entities that can be substituted for occurrences of the ordinary variables  $z$  and  $st$ , respectively. Specifically, if the substitution is to be valid, the value of the attribute StarshipZone for any Starship instance substituted for  $st$  must be equal to the Zone instance substituted for  $z$ . The only way of enforcing this restriction in the current version of Quiddity\*Suite is by encoding it procedurally in the entity creation process. In other words, it is not possible to express this kind of restriction in the frame definition alone, so we have to enforce it via the entity creation process, or A-box construction procedure.

This limitation in expressing context nodes in Quiddity\*Suite is akin to the DL limitation discussed in the end of Subsection 2.4.1 concerning the representation of constraints on the instances that can participate in a relationship. That is, just as we

cannot express certain constraints using only the T-Box in a DL representation, we cannot represent those same constraints in the frame structure of a Quiddity\*Suite model.

The next version of Quiddity\*Suite will have the ability to use Prolog-like rules for automatically controlling A-box construction to enforce the restrictions expressed in the context nodes of an MFrag. Unfortunately, the new version will not be released in time to be incorporated into this research. As a result, we enforced context restrictions procedurally in our model's function definitions and executable module, both available in Appendix A.

As a general rule, context nodes restricting the conditions under which the instances of an MFrag should be created can only be expressed in Quiddity\*Suite in a programmatically fashion during the creation of the instances. The inclusion of logical rules will allow a modeler to define the rules prior to the A-box creation process, thus introducing a great level of automation in a procedure that we had to perform by carefully programming the A-box creation itself.

The last specific translation issue to be addressed when translating MTheories to Quiddity\*Suite is the case in which a node has many possible parents. One example of such situation is illustrated in Figure 26, where the resident node  $ZoneMD(z, t)$  has the input node  $CloakMode(st)$  as its parent and the restrictions expressed in the context nodes allow this input to have many possible instances. In other words, given the valid context for the Zone MFrag, all starships that happen to be in zone  $z$  will be parents of  $ZoneMD(z, t)$ . Thus, we have a variable number of parents that depends on how many starships are in a given zone.

The problem is that Quiddity\*Suite doesn't have support for defining probability distributions for an undefined number of parents, so specifying a distribution such as the one we have shown in Figure 8 is not possible at this time. Therefore, in order to model this kind of situation we had to resort to a modeling trick in which we created a "collector node" in Zone MFrag and used Quiddity's MaxDistribution to handle the undefined number of parents.

In frame Zone, the slot pointing to frame Starship is the starship slot. Usually, an external parent of slot zoneMD would be listed using the pointer.parent format, such as starship.cloakMode in this case. However, we created an intermediate node called anyStInCloakMode which has starship.cloakMode as a parent. This node has the MaxDistribution as its probability distribution so it can handle multiple parents. Then, the "collector" node anyStInCloakMode, and internal node to the Zone MFrag is listed as a parent of zoneMD slot. The intermediate node was necessary because the MaxDistribution accepts only one parent in its list, and ZoneMD has more than one parent so it wouldn't work with that distribution.

As a general rule, if the variable number of parents of a resident node is generated from one input node only and this is its only parent, then it is possible to use the MaxDistribution directly (i.e. use the same rules for input parents defined in Subsection 4.2.1). Else, the scheme of an intermediate, "collector" node is necessary. In any case, only special distributions such as the MaxDistribution are allowed. In the Starship model, the MaxDistribution was used in place of the one we defined in the pseudo-code of Figure 8.

### 4.2.3 Use of Comments and Other Aspects of Quiddity\*Suite

In order to facilitate the translation between MEBN representation and Quiddity\*Suite Models, we have developed a list of markups that should be included as comments in Quiddity\*Suite models. The information inside each markup would then be inserted in specific concepts in a PR-OWL ontology. Table 4 brings a list of those markups and their respective PR-OWL concepts.

Table 4. Metadata Annotation Fields

<b>Markup Label</b>	<b>PR-OWL Concept</b>
MEBNUID	UID (datatype property of entities).
NodeType	Subclass of PR-OWL class Node that a given RV belongs to.
NodeMFragment	Lists the individual of PR-OWL class MFragment that a given node belongs to.
NodeHomeMFragment	If a resident node, then this field will have the same value as above. If an input node, then it will list the MFragment were its distribution is defined.
NodeDistType	Lists the format being used to define the distribution.
NodeDescription	This field should be transposed to the annotation field of the PR-OWL ontology.
NodeDist	If a table, it reads: “see table”. Else, it’s the field that should convey the probability distribution formulas. In PR-OWL is the datatype property hasDist (of declarative distributions).
NodeDistComments	This field should be transposed to the annotations of the probability distribution individual. Basically, it explains the intended meaning/rationale of a given probability distribution.
QuiddityName	This field handles the naming differences between the PR-OWL ontology, Netica, and Quiddity models. In PR-OWL ontologies, it should be listed in the annotations field.
QuiddityObj	Another translation facilitator. It describes whether a MEBN entity is a frame, slot, or facet in a Quiddity model.

Quiddity\*Suite has been applied to many complex, real-world problems (e.g., Alghamdi *et al.*, 2004; Fung *et al.*, 2004; e.g., Alghamdi *et al.*, 2005; Costa *et al.*, 2005). Its powerful representation and reasoning capabilities have provided solutions to problems that could not be solved with previously existing technology. Development of Quiddity\*Suite is ongoing, and new capabilities are being added on a continuing basis. As of the final phase of writing this dissertation, a new version is being released that incorporates significant advances in the use of Prolog rules to establish constraints in the slot instantiation process.

Those advances allow Quiddity models to replicate the context nodes of an MFrag, adding a major capability that as far as our knowledge goes is not implemented in any similar system. The time frame of this work prevent us to perform a more detailed analysis to evaluate the impact of those advances, and to verify whether full compatibility with MEBN logic has been achieved. Yet, the fact that we were able to built three logically equivalent versions of the Star Trek model using Quiddity\*Suite, MEBN logic, and PR-OWL is a clear indication that in its current stage, Quiddity\*Suite can be used as a reasoner in MEBN-based probabilistic ontologies.

The set of rules we have just described, combined with the extended version of MEBN logic we presented in the previous section, represent our solution for the two major issues preventing the development of a probabilistic ontology language. In the next chapter, we show how we have built upon what we implemented in “clearing the path” for probabilistic ontologies to develop PR-OWL, a probabilistic extension to the OWL Web ontology language.



## Chapter 5 PR-OWL

As a means to realize the use of Bayesian theory for representing and reasoning under uncertainty in the Semantic Web, this Chapter proposes a standard knowledge representation formalism to express uncertain phenomena, performing plausible reasoning, and learning from data in the context of the Semantic Web. This formalism will provide a framework for executing those tasks in an interoperable way, so probabilistic ontologies that were built for different purposes, using diverse tools, and by knowledge engineers that were not mutually aware of each other's work, would have a common underlying architecture guaranteeing the exchange of information in a useful and meaningful way.

A framework intended to provide means for building probabilistic ontologies for the Semantic Web must be compatible with the technologies being used in that environment. Thus, since OWL is the recognized ontology language of choice for the Semantic Web, it is also our base language for building the framework for probabilistic ontologies. That is, PR-OWL is an extension of OWL that enables the specification of probabilistic ontologies.

The OWL Web ontology language is a W3C Recommendation, which means it is the product of an exhaustive, consensus-based process in which many highly qualified participants from different countries composed various working groups and generated the

technical reports which collectively comprise the final Recommendation (c.f. Jacobs, 2003). Extending a W3C Recommendation requires a similar process and implies a level of commitment from the W3C that makes it clearly outside the scope of a PhD Dissertation. Still, as explained in the previous chapters, the W3C's vision for the Semantic Web can only be achieved with a sound and principled treatment of inconclusive, ambiguous, incomplete, unreliable, and dissonant data, all quite abundant in the current World Wide Web environment.

We saw in Chapter 3 that Bayesian probability theory provides a means for representing, reasoning and learning from all the above cited varieties of uncertain data, and is thus a natural candidate for providing the much-needed probabilistic framework for the Semantic Web. The development of a strategy for building that framework can be embraced as a doctoral research effort, and that is precisely the intention of the present work and the main focus of this chapter. Furthermore, the present work is envisioned as a basis for incorporating uncertainty in a future version of OWL.

The Chapter is divided into two main sections. The initial section establishes an implementation strategy for PR-OWL, which includes further considerations on probabilistic ontologies, the reasons for choosing MEBN logic as the underlying semantics of PR-OWL, and the intended scope of its definitions. The second part of the Chapter presents PR-OWL itself, and covers the major characteristics of the language.

## 5.1 The Overall Implementation Strategy

Before devising a way of implementing a framework for building interoperable probabilistic ontologies, it is important to emphasize that a probabilistic ontology is not a probabilistic model (e.g. a model built using applications such as Netica, Hugin, or Quiddity\*Suite) the same way that an ontology is not a database application.

The differences in the in-depth underlying concepts and technologies supporting ontologies and database schemas are not easily distinguishable, as the real differentiation between the two resides in their respective intended purposes. Ontologies represent domains in a way that should facilitate interoperability with other representations of that domain (i.e. other ontologies build by different people with different views and interests) or of domains that are not directly related but share some concepts. When a database solution for a given domain is conceived, its primary focus is not in representing all concepts of a domain in a way that makes it interoperable with current or future views of that domain, but in defining the concepts of that domain which would allow to coherently store the information the database stakeholders (and their customers) want to store and to retrieve that information in a way that best fits their requirements.

In a similar view, when a probabilistic model is built to solve (say) a radar data fusion problem, the main interest driving its creators is not in making sure that their definitions about radar domain concepts are interoperable with other definitions that might exist on those same concepts. In contrast, interoperability would definitely be a primary focus when building a probabilistic ontology for the domain of radar data fusion. Ontology engineers would attempt to express one view of that domain in a way that

others (with possibly different views) may use/understand and thus build applications (databases, decision systems, etc) that are compatible with anything built under that view.

Furthermore, it is not necessary for an ontology to be an actually running database, yet a database application can be built on top of an ontology. Likewise, a probabilistic ontology does not necessarily need to be an actually running probabilistic model, yet a running probabilistic model (i.e. an executable application built using a probabilistic package) can be built on top of a probabilistic ontology if that fits the objectives of the application at hand. A subtle difference here is that anything built on top of an ontology can be built on top of a probabilistic ontology, but the converse is not always true, since the latter is an extension of the former that adds the above mentioned features of a probabilistic framework.

To comply with interoperability requirements and at the same time be useful enough for allowing a probabilistic model to be built on top of its definitions, a probabilistic ontology has to be based on a very flexible framework. Thus, the initial issue to be addressed is the definition of an underlying model for PR-OWL, one that allows representing uncertain data using OWL's RDF based syntax. Clearly, it is desirable that the semantics of such model should have at least the same representational power of the semantics supporting OWL, so everything that can be represented in OWL could also be expressed in PR-OWL.

#### 5.1.1 Why MEBN as the semantic basis for PR-OWL?

In general, people faced with the complex challenge of representing uncertainty in languages like OWL tend to start their attempts by writing probabilities (i.e. priors and

CPTs) as annotations (e.g. marked-up text describing some details related to a specific object or property). This is a palliative solution that addresses only part of the information that needs to be represented, since it fails to convey the structural intricacies that are present in even the simplest probabilistic models, such as conditional dependence (or independence) implied by connecting arcs (or lack of), double counting of influence on multiply connected graphs, and others.

Indeed, many researchers have pointed out the importance of structural information in probabilistic models (e.g. Shafer, 1986; Schum, 1994; Kadane & Schum, 1996). For instance, Schum (1994, page 271) shows that some questions about evidence can be answered entirely in structural terms.

In short, annotating the numerical probabilities of a probabilistic model in an ontology is just not enough, as too much information is lost to the lack of a good representational scheme that captures the structural nuances of the model. As noted in Chapter 2, one way of representing structural information of a probabilistic model is by extending OWL to represent Bayesian networks (e.g. Ding & Peng, 2004). However, even though such approach does capture some of the structural information of a probabilistic model, the limited expressiveness of Bayesian networks make it difficult to represent complex systems, as we could see from the Starship example in Chapter 3.

Probabilistic Relational Models provide a leap in representation power when compared with BNs, but as we could see in Chapter 4, PRMs alone cannot represent all that is needed for declarative representations that cover complex situations with tightly defined contexts (i.e. situations in which probability distributions are defined within very

specific constrains). Therefore, to represent one of those specific situations in an ontology using PRMs as the underlying logic, either the instances of that ontology would also have to be declared (e.g. expressing that two starships are not the same individual by referring to two actual instances of starships<sup>25</sup>) or some combined approach would have to be used for constraining the context where the definitions apply (e.g. the use of Prolog rules in Quiddity\*Suite).

The need to declare all instances in advance makes the first solution unsuitable for most use cases for the Semantic Web, where the ontologies generally have only T-Box information (or occasionally a few built-in A-Box definitions) and the A-Box is left for each specific situation/application based on that ontology. Thus, the second solution seems to be a more appropriate way of employing PRMs to build probabilistic ontologies. One successful example of that approach is the use of Prolog rules in Quiddity\*Suite as a means to enforce constraints under which instances of a random variable are created.

Establishing such constraints is a vital asset for building probabilistic ontologies, since it allows one to express very detailed situations in which a given probability distribution holds. MEBN logic has a built-in form of representing such constraints (i.e. its context nodes), which makes it a flexible and simple technology that is also logically coherent (i.e. it can express a fully coherent joint probability distribution over instances that satisfy the constraints). These intrinsic features of MEBN logic makes it very suitable for being the basis of a probabilistic framework for the Semantic Web.

---

<sup>25</sup> Refer to the examples and discussions in the end of subsections 2.4.1 and 4.2.2

Approaches with limited expressiveness, such as BNs, are less suitable because the Semantic Web demands a certain level of flexibility those approaches cannot deliver. More expressive representational schemes such as PRMs, implementations of MEBN logic, and probabilistic logic programs theoretically have the basic conditions for supporting such a framework. In any case, there will always be a trade-off between flexibility and expressiveness when using a probabilistic logic to support a language meant for the Semantic Web. We found that MEBN logic provides a particularly attractive trade-off that made our work easier when extending the OWL Semantic Web language.

Laskey (2005, pages 22-27) shows that MEBN logic can express a joint probability distribution over models of any consistent finitely axiomatizable theory in classical first order logic. Thus, even the most specific situations can be represented in MEBN, provided they can be represented in FOL. Furthermore, since MEBN is a first order Bayesian logic, using it as the underlying semantics of PR-OWL not only guarantees a formal mathematical background for a probabilistic extension to the OWL language (PR-OWL), but also ensures that the advantages of Bayesian Inference (e.g. natural “Occam’s Razor”, support for learning from data, etc.) will be available for using with any PR-OWL probabilistic ontology.

Therefore, we opted to use MEBN logic as the underlying semantics of OWL for its optimal combination of expressiveness and flexibility. Our next step is to lay out an overall plan for implementing it in a way that does not render current OWL ontologies incompatible with the extended language.

### 5.1.2 Implementation Approach

OWL has intrinsic mechanisms to enable the development of extensions. The most basic means of extending OWL is to specify a vocabulary using a syntax that complies with its format. As an example, the Dublin Core metadata initiative is devoted to developing specialized metadata vocabularies and to promote the widespread adoption of interoperable metadata standards (Hillmann, 2001). Any OWL ontology can use the Dublin Core vocabulary to define additional semantics about its contents just by adding its namespace in the file header and encoding qualified Dublin Core Metadata in the RDF / XML format described in Kokkelink & Schwänzl (2001).

Yet, this basic level of extensibility is not enough to guarantee a coherent, widely used standard for more complex activities that demand a greater level of commitment from users of the standard. As an example, OWL-S (Martin *et al.*, 2004) and the Web Service Modeling Ontology – WSMO (Polleres *et al.*, 2005) have been acknowledged as Member Submissions, both proposing solutions for Web service ontologies. Their respective sponsoring organizations submitted the draft specifications with the hop that these can form the basis of a future standard and thus form a framework that would allow a much higher degree of automation, functionality and interoperability among the various types of services. Those specifications build upon and extend the foundation laid by OWL and other web standards; PR-OWL as described below intends to make a similar contribution.

Some extensions do change the semantics and abstract syntax of OWL. As an example, the Semantic Web Rule Language (SWRL) is a W3C Member Submission that



proposes to extend OWL abstract syntax so it includes support for rules based on RuleML (Boley & Tabet, 2004) and provides a model-theoretic semantics defining the meaning of the rules written in the extended syntax (Horrocks *et al.*, 2004). In addition, there is another W3C Member Submission that proposes extending SWRL so it would allow OWL ontologies containing the extended abstract syntax and semantics defined in SWRL to handle unary/binary first-order logic (Patel-Schneider, 2005).

The extensions listed in the above paragraph do add new elements to the abstract syntax and semantics of OWL, which means they augment the expressiveness of the language by enabling it to express concepts that are not possible to convey with standard OWL. On the other side, in order to make use of those extensions, it is necessary to develop new tools supporting the extended syntax and implied semantics of each extension.

PR-OWL is an extension that enables OWL ontologies to represent complex Bayesian probabilistic models in a way that is flexible enough to be used by diverse Bayesian probabilistic tools (e.g. Netica, Hugin, Quiddity\*Suite, JavaBayes, etc.) based on different probabilistic technologies (e.g. PRMs, BNs, etc.).

That level of flexibility can only be achieved using the underlying semantics of first-order Bayesian logic, which is not a part of the standard OWL semantics and abstract syntax. Therefore, it seems clear that PR-OWL can only be realized via extending the semantics and abstract syntax of OWL the same way as the above examples of SWRL, RuleML and SWRL-FOL.

Indeed, an ideal full implementation of a probabilistic ontology would follow the steps defined by the W3C (Jacobs – ed., 2003) until it becomes an official standard. As demonstrated in Chapter 3, all the information needed to process probabilistic queries in a MEBN models is contained in the model’s generative MTheory and the findings related to the query of interest. Also, we have shown that one of the advantages of MEBN logic is the ability to express very specific situations via context nodes, which are declarative statements with FOL expressiveness.

Therefore, and that constitutes one of the major contributions of the present work, it is possible to define an upper ontology for probabilistic systems that can be used as a framework for developing probabilistic ontologies (as defined in the beginning of this Chapter) that are expressive enough to represent even the most complex probabilistic models.

Defining such a framework is the initial step towards a full PR-OWL specification, and a basic requirement for the development of probabilistic ontologies. With that in mind, the implementation strategy that guided our actions in the present research effort consisted of the following steps:

- a. Define the formal foundation (based on Bayesian first-order logic) needed to specify general probabilistic ontologies.
- b. Present an operational concept to provide a general guidance on the development of plug-ins and/or applications that make it easier for the average user to write probabilistic ontologies.

- c. As a step towards standardization by the W3C, establish a future vision for the PR-OWL specification, and a plan for realizing that vision.

Steps “a” and “b” are covered in the remainder of this Chapter, while the last step is addressed in Chapter 6.

## 5.2 An Upper Ontology for Probabilistic Systems

Our initial step towards a Bayesian framework for the Semantic Web is to create an upper ontology to guide the development of probabilistic ontologies. DaConta *et al.* define an upper ontology as a set of integrated ontologies that characterizes a set of basic commonsense knowledge notions (2003, page 230). In this preliminary work on PR-OWL as an upper ontology, these basic commonsense notions are related to representing uncertainty in a principled way using OWL syntax. If PR-OWL were to become a W3C Recommendation, this collection of notions would be formally incorporated into the OWL language as a set of constructs that can be employed to build probabilistic ontologies.

The PR-OWL upper ontology for probabilistic systems is presented in Appendix B. It consists of a set of classes, subclasses and properties that collectively form a framework for building probabilistic ontologies. The first step toward building a probabilistic ontology in compliance with our Definition 3 (pages 101/102) is to import into any OWL editor an OWL file containing the PR-OWL classes, subclasses, and properties. In fact, this is exactly what we did when we built the Star Trek probabilistic

ontology. We used the Protégé import feature to download the PR-OWL upper ontology from a website we had previously set up.

After importing the PR-OWL definitions, the next step in ontology design is to construct domain-specific concepts, using the PR-OWL definitions to represent uncertainty about their attributes and relationships. As an example, the concepts of the Star Trek probabilistic ontology were either subclasses or instances of the imported PR-OWL upper ontology. Using this procedure, an ontology engineer is not only able to build a coherent generative MTheory and other probabilistic ontology elements, but also make it compatible with other ontologies that use PR-OWL concepts.

Because we designed PR-OWL with the objective of eventually turning it into a W3C submission, we wanted it to be as general purpose as possible. That is, we attempted to avoid unnecessary restrictions that would initially make the job easier for the designer of a specific application, but would limit its flexibility for a broader set of applications. Imposing such limitations would render this preliminary work less suitable as the starting point for a W3C Recommendation process. Thus, even though we did establish a fixed set of classes, subclasses and instances for the upper ontology, which was necessary to enforce consistency with MEBN logic standards, we intentionally avoided unnecessary restrictions on how a modeler would develop her/his own probabilistic ontology. It is clear to us that such approach is valid for the scope of this work, but the natural tendency for the process towards a W3C Recommendation is to impose extra restrictions that would achieve an optimal trade-off between flexibility and enforcing the rules of the underlying logic. In other words, an upper ontology is enough

as a starting point to represent uncertainty in a principled way using PR-OWL, but it cannot prevent unintentional misuse of its elements that would lead to inconsistencies in the resulting probabilistic ontology.

In order to illustrate our conceptual approach, consider the question of whether to represent an MFrag template such as the Zone MFrag from our Star Trek generative MTheory (see Figure 10, page 70) as a class or an instance. If we choose the first option, we would create it as a subclass of the imported PR-OWL class Domain MFrag (see Appendix B, page 230). That newly created subclass will thus inherit all the properties from the PR-OWL Domain MFrag class that enforce the structural and logical constraints of a MEBN Fragment (e.g. it must have at least one resident node, it might have context and input nodes, etc.). The instances of that subclass would then be copies of the Zone MFrag template that have all of its inherited elements. This approach seems appropriate when the ontology being built is supposed to represent the many copies of Zone MFrag created by SSBN construction procedures started to answer a given query.

If, instead, we opt for the second approach and represent the Zone MFrag template as a direct instance of the PR-OWL class Domain MFrag, then such instance would still carry all the properties of a Domain MFrag (e.g. it must have at least one resident node, etc.) that enforce the structural and logical constraints of MEBN logic. In this case, unless we want to use a second order representation (i.e. use instances of instances), the ontology itself could not contain instances of the Zone MFrag template. We could add instances of the random variables that appear in the Zone MFrag (e.g., ZoneMD(!Z0)) to the ontology, but there would be no instance of the MFrag template

explicitly represented in the ontology. The PR-OWL instance we created to represent the Zone MFrag template would contain all the information that an application external to the ontology (i.e. a decision support system) needs to build copies of Zone MFrag when building SSBNs to answer a query.

It is important to keep in mind that no matter what approach an ontology designer uses in the light of his/her objectives, the structural and logical constraints of MEBN logic will be inherited. Since the other elements of the “probabilistic part” of the ontology will also be either instances or subclasses of the imported PR-OWL upper ontology, then all will inherit the structural and logical constraints that collectively enforce the compliance with MEBN rules, thus guaranteeing that such an ontology would be a coherent, logically consistent MEBN Theory.

Although we did not establish any constraints on this specific issue, we considered the pros and cons of modeling our concepts as subclasses or instances of PR-OWL classes in the design of our Star Trek probabilistic ontology. Our experience leads us to conclude that the objectives and characteristics of the probabilistic ontology being built will dictate how to make this choice. In general, ontologies that are expected to represent many instances of a given concept (e.g. copies of Zone MFrag in the illustration above) should characterize that concept as a subclass of PR-OWL. Conversely, if a given concept is not going to have its instances represented in the ontology (e.g. only the Zone MFrag template is of interest) then the concept itself might be characterized as an instance of a PR-OWL class. The advantage of doing so is to avoid unnecessary duplications (e.g. many copies of a Zone MFrag template that would not be used by the

client applications of the Star Trek ontology). In the Star Trek probabilistic ontology, most of the concepts directly related to the generative MTheory were modeled as instances, whereas Object entities such as Starships and other concepts for which we expect to have its instances populating the ontology were modeled as classes.

Our choice took into account that representing uncertainty within an ontology is not the same thing as building a probabilistic system. In our Star Trek case study, the generative MTheory is used in conjunction with information about domain entities (e.g. instances of starships) to build SSBNs to answer queries about those entities. In this case, the Enterprise's decision support system would carry out the process of building situation-specific models (i.e. instantiating and combining MFragments) to answer the relevant queries, evaluate the perceived situation, and update the system's knowledge accordingly. The generative MTheory can be seen as the part of the system that holds the domain knowledge used in this process. In other words, the process of building, working and storing the instantiated MFragments in this case is not part of the Star Trek probabilistic ontology.

Even though we understand the above option might be desirable in some applications, we preferred to adopt a different approach that avoids duplications by restricting the user defined classes only to the elements we expect to be instantiated in the ontology itself (as distinct from an application that uses the ontology). In short, we opted to represent the generative MTheory concepts as instances of PR-OWL built-in classes, while representing the object entities, random variables (i.e. resident nodes), and its distributions as user defined classes. As an example, Starship would be a user-defined

class (subclass of PR-OWL ObjectEntity class) whose instances would be something such as !*ST0*, !*ST1*, etc., whereas the Zone MFrag is modeled as an instance of PR-OWL built-in Domain MFrag. This approach is consistent with the fact that a generative MTheory contains all the domain-specific information that is needed in conjunction with information on the object entities for the targeted application of our ontology to conduct its reasoning processes (e.g. the Enterprise's decision support system). In the end, we believed our choice to be preferable in most cases in which an ontology is needed, because it results in a more concise ontology that still can be used for applications as the basis for conducting their respective reasoning process.

A generative MTheory can express domain-specific ontologies that capture statistical regularities in a particular domain of application, and MTheories with findings can augment statistical information with particular facts germane to a given reasoning problem (Laskey, 2005). From our definition, it is possible to realize that nothing prevents a probabilistic ontology from being “partially probabilistic”. That is, a knowledge engineer can choose the concepts that he/she is interested to be in the “probabilistic part” of the ontology, while writing the other concepts in standard OWL.

In this specific case, the “probabilistic part” refers to the concepts written using PR-OWL definitions and that collectively form an MTheory. There is no need for all the concepts in a probabilistic ontology to be probabilistic, but at least some have to form a valid MTheory. Of course, only the concepts being part of the MTheory will be subject to the advantages of the probabilistic ontology over a deterministic one.



The subtlety here is that legacy OWL ontologies can be upgraded to probabilistic ontologies only with respect to the concepts for which the modeler wants to have uncertainty represented in a principled manner, make plausible inferences from that uncertain evidence, or to learn its parameters from incoming data using Bayesian learning.

The ability to perform probabilistic reasoning with incomplete or uncertain information conveyed through an ontology is a major advantage of PR-OWL. However, it should be noted that in some cases solving a probabilistic query might be intractable or even undecidable. In fact, providing the means to ensure decidability was the reason why the W3C defined three different version of the OWL language. While OWL Full is more expressive, it enables an ontology to represent knowledge that can lead to undecidable queries. OWL-DL imposes some restrictions to OWL in order to eliminate these cases. Similarly, restrictions of PR-OWL could be developed that limit expressivity to avoid undecidable queries or guarantee tractability. This initial step is focused on the most expressive version of PR-OWL.

In this section, the “probabilistic part” of PR-OWL ontologies will be covered, and the main objective is to show how to represent any generative MTheory (with no regard to its level of complexity) and also Finding MFragments using PR-OWL concepts. An overview of the general concepts involved in the definition of an MTheory in PR-OWL is depicted in Figure 27. In this diagram, the ovals represent general classes, while the major relationship between those classes are symbolized by arrows. A probabilistic ontology has to have at least one individual of class MTheory, which is basically a label

linking a group of MFrag that collectively form a valid MTheory. In actual PR-OLW syntax, that link is expressed via the object property hasMFrag (which is the inverse of object property isMFragIn).

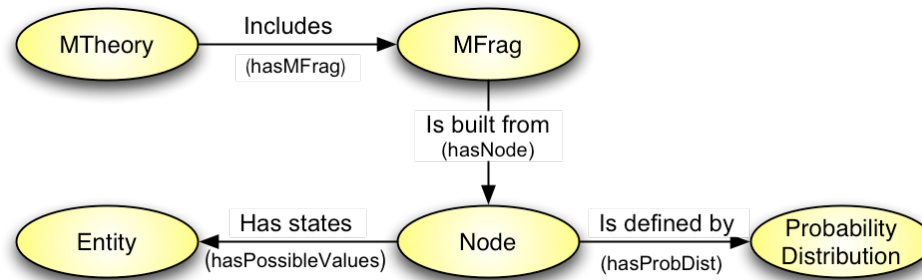


Figure 27. Overview of a PR-OWL MTheory Concepts

Individuals of class MFrag are comprised of nodes, which can be resident, input, or context nodes (not shown in the picture). Each individual of class Node is a random variable and thus has a mutually comprehensive, 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 Nodes in PR-OWL) have unconditional or conditional probability distributions, which are represented by class Probability Distribution and linked to its respective nodes via the object property hasProbDist.

The scheme in Figure 27 is intended to present just a general view and thus fails to show many of the intricacies of an actual PR-OWL representation of an MTheory. Figure 28 shows an expanded version conveying the main elements in Figure 27, its subclasses, the secondary elements that are needed for representing an MTheory and the reified relationships that were necessary for expressing the complex structure of a Bayesian probabilistic model using OWL syntax.

Reification of relationships in PR-OWL is necessary because of the fact that properties in OWL are binary relations (i.e. link two individuals or an individual and a value), while many of the relations in a probabilistic model include more than one individual (i.e. N-ary relations). The use of reification for representing N-ary relations on the Semantic Web is covered by a working draft from the W3C's Semantic Web Best Practices Working Group (Noy & Rector, 2004).

Although the scheme in Figure 28 shows all the elements that are needed for representing a complete MTheory, it is clear that any attempt at a complete description would render the diagram cluttered and incomprehensible. Therefore, a complete account of the classes, properties and the code of PR-OWL are given in Appendix B

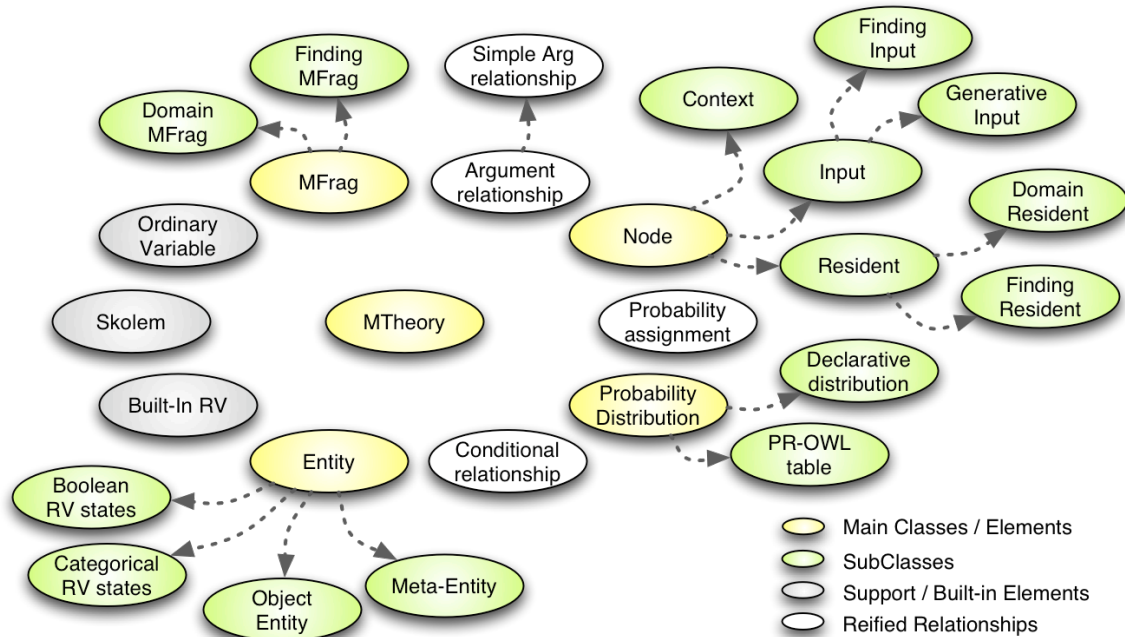


Figure 28. Elements of a PR-OWL Probabilistic Ontology

The material provided in the appendix defines an upper ontology for probabilistic systems, and it can be used to represent any system that can be represented using the

extended version of MEBN logic presented in Chapter 4. In order to show the applicability of the presented framework, the next Subsections explain how it can be used to build a probabilistic ontology.

In order to demonstrate the applicability of PR-OWL in diverse levels of complexity, initially a generic explanation is given for each major aspect of the modeling process, then an illustrative example based on the Starship case study is provided as a means to facilitate the understanding over the most important steps. In both cases, the examples were built using the open source software Protégé<sup>26</sup>, an ontology editor developed by the by Stanford Medical Informatics at the Stanford University School of Medicine (Noy *et al.*, 2000; Noy *et al.*, 2001), and its OWL plugin (Knublauch *et al.*, 2004).

At the present experimental stage, writing probabilistic ontologies in PR-OWL is a process that requires importing the upper ontology provided in Section B.4 in the appendices. Figure 29 shows the header of the Starship probabilistic ontology developed as a case study for this research. There, it is possible to see the owl:imports feature being used for downloading the PR-OWL upper ontology utilized as the base block for building the Starship probabilistic ontology.

Even though the above example was written in Protégé, any ontology tool capable of editing OWL ontologies, such as SWOOP<sup>27</sup> (Kalyanpur *et al.*, 2004) or webODE<sup>28</sup> (Arpírez *et al.*, 2001), can be used for editing a PR-OWL ontology.

---

<sup>26</sup> Available for download at <http://protege.stanford.edu/>

<sup>27</sup> Available for download at <http://www.mindswap.org/2004/SWOOP/>

<sup>28</sup> See <http://webode.dia.fi.upm.es/WebODEWeb/index.html>

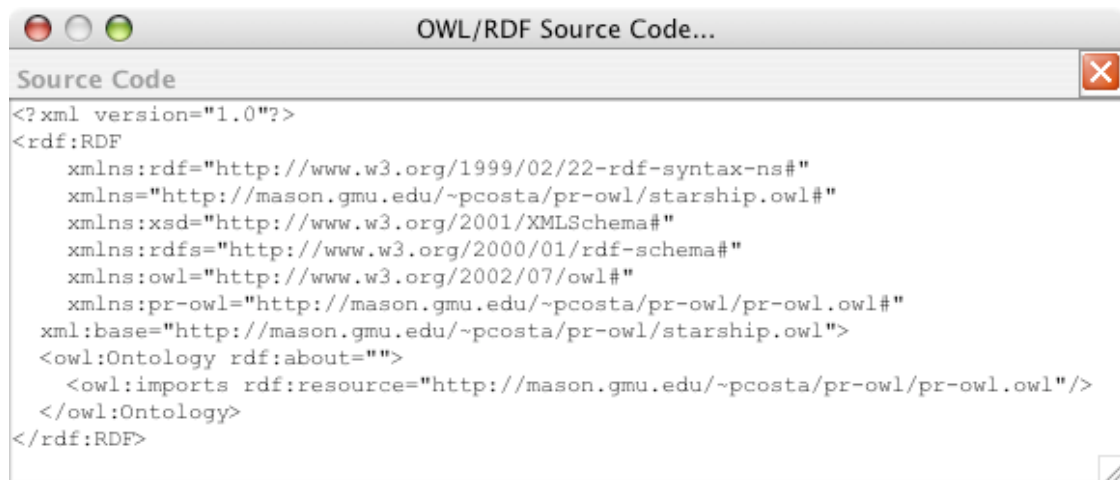


Figure 29. Header of the Starship Probabilistic Ontology

### 5.2.1 Creating an MFrag

Figure 30 shows the initial Protégé screen after importing the PR-OWL ontologies and defining the classes of object entities that will be part of the ontology. In Protégé, concepts of imported ontologies appear with a light colored dot icon and the namespace abbreviation at the left side of the concept's name, as it can be seen in the Asserted Hierarchy window on the left side of the picture.

The darker icons (Starship, Zone, Sensor Report, and TimeStep) correspond to the classes created as a first step to build the Starship probabilistic ontology. PR-OWL object entities correspond to frames in frame systems and to objects in object-oriented systems. The simple model used in this research contains only four object entities; so four classes were created under the PR-OWL ObjectEntity Class (i.e. Starship, Zone, SensorReport, and TimeStep). These are the user-defined classes that convey the equivalent of what a standard ontology would represent about a domain, so its individuals are the concepts and entities that would populate a non-probabilistic description of that domain. In our Starship ontology, the domain instances will be individual zones, sensor reports,

starships, and time steps, all represented as individuals of the domain classes created by the user.

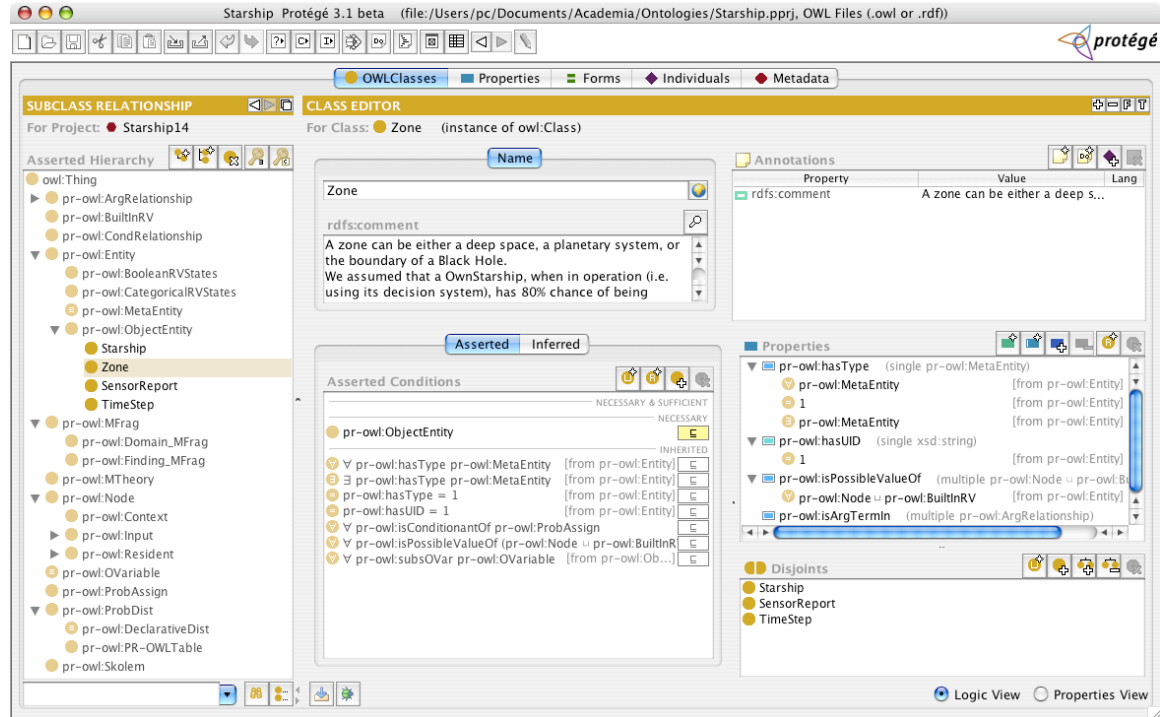


Figure 30. Initial Starship Screen with Object Properties Defined

The other PR-OWL classes shown in the picture are directly fulfilled by individuals representing the elements of a generative MTheory. The user does not create new classes here, but individuals that convey the information necessary for creating elements of an SSBN. In other words, these individuals express the probabilistic aspects of the domain MTheory, and can be seen as templates that a probabilistic reasoner uses for building an SSBN to answer a query. Examples of those aspects are the characteristics of domain instances (e.g. the possible nature of a zone, class of a starship, etc), its possible states, its probability distributions, etc.

When Quiddity\*Suite (or another probabilistic reasoner chosen by the user) receives a query on (say) the status of zones !Z0, !Z1 and !Z8 (all individuals of the user-defined domain class Zone) it will build an SSBN based on the individuals of PR-OWL classes representing the generative MTheory and the evidence available in form of findings. In this case, the reasoner will certainly build three copies of RV *ZoneNature*(z) based on the information contained in the individual *Z\_ZoneNature* of the PR-OWL class *Domain\_res*.

Even though the names chosen for the four object entity classes match their respective intended meaning, this is not a requirement. PR-OWL uses a UID as a means to enforce its unique naming assumption, and the name of each concept has no meaning for the logic under PR-OWL (MEBN logic). As an example, choosing a name such as “Umbrella” as the reference to the class including all sensor reports in the model would make no difference for the tasks performed by the reasoner, but would certainly confuse any human reader trying to understand the model. Therefore, as a means to facilitate human understanding and to improve interoperability with other systems (which probably have humans as API builders), an optional naming convention is proposed in Section B.3 in the appendices and was used in the Starship probabilistic ontology built for this research.

Figure 31 illustrates the PR-OWL representation of the Zone MFrag, which uses the above-cited naming convention. An MFrag can be seen as a hub connecting a collection of related random variables that together represent an atomic "piece of knowledge" about a domain. The context nodes of the MFrag represent conditions under

which the relationship holds. A coherent set of those “pieces” form a joint probability distribution over the included random variables, also known as an MTheory.

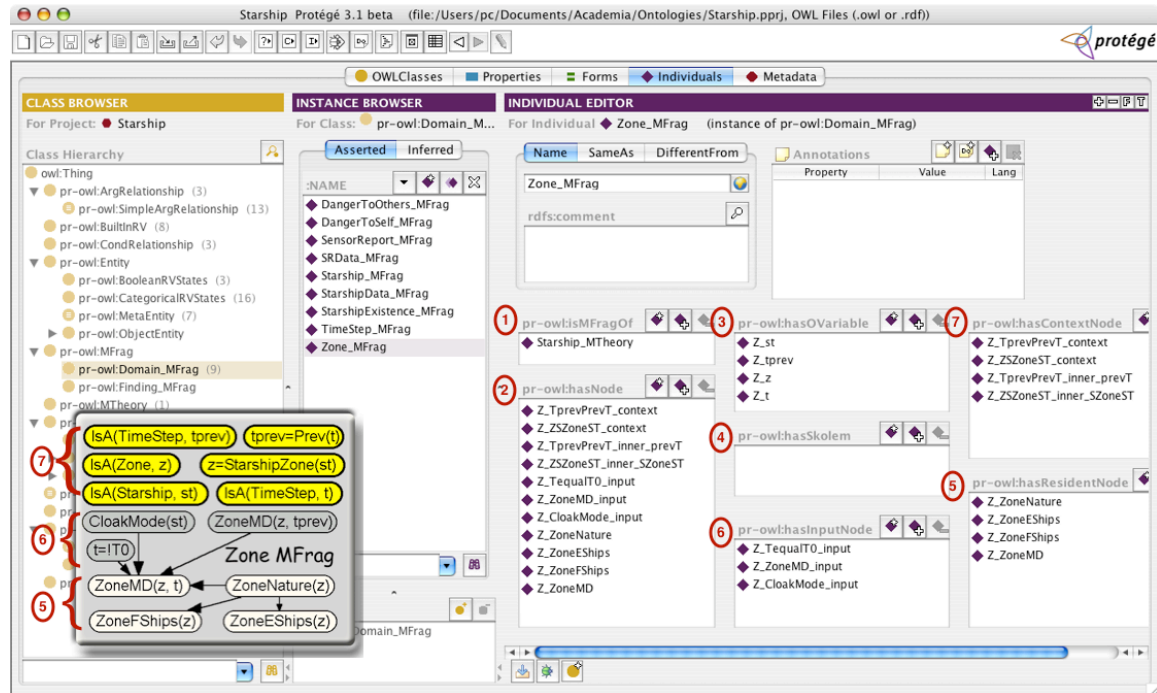


Figure 31. Zone MFrag Represented in PR-OWL

A common method for handling cognitive tasks is the “divide and conquer” approach, which breaks a problem into smaller, simpler parts. Thus, building “pieces of knowledge” about a domain in a way that allows “gluing” them together to handle more complex issues within that domain is a natural technique for modeling probabilistic systems. Not surprisingly, a very usual way of starting a probabilistic ontology is by defining its generative MFrag or, in PR-OWL, the individuals of class `Domain_MFrag`.

PR-OWL includes all the necessary elements of MEBN logic that are necessary to represent an MFrag. Figure 31 shows MEBN’s representation of the Zone MFrag in comparison with its PR-OWL counterpart. Following the bullets within the figure, every



individual of class `Domain_MFrag` is related to one or more `MTheory` via the object property `isMFragOf [1]`<sup>29</sup>.

The Zone MFrag is represented as an individual of class `Domain_MFrag`, having name `Zone_MFrag`. The MFrag has four resident nodes, three input nodes and six context nodes. Its PR-OWL represents those 13 nodes using 11 individuals of subclasses of `Node`, which are linked to the `Zone_MFrag` via the object property `hasNode [2]`. The mismatch between the number of MEBN nodes and their respective PR-OWL description is caused by the fact there is not a straightforward one-to-one correspondence between MEBN and PR-OWL constructs. Table 5 shows the details of how each node is portrayed in both representations.

As shown in the table, the “IsA” context nodes are not explicit represented in PR-OWL MFrag, since the notion of subtyping is already conveyed in the definition of the arguments of each resident node. In MEBN, the “IsA” context nodes are meant to define which type of entities can substitute the ordinary variables in an MFrag. In PR-OWL, this constraint is expressed by the object property `isSubsBy`, linking individuals of class `OVariable` to the individuals of class `Entity` that are allowed to substitute for them. As an example, Zone MFrag has four ordinary variables (*st*, *t*, *tprev*, and *z*) that are represented in PR-OWL as four individuals of class `OVariable` (`Z_st`, `Z_t`, `Z_tprev`, and `Z_z`). Thus, while in MEBN logic the context node `IsA(Starship, st)` is meant to restrict the ordinary variable *st* so that only entities of type `Starship` can substitute for it, in PR-OWL the

---

<sup>29</sup> Numbers inside brackets refer to the equally numbered circle labels in the pictures

equivalent construction is  $\text{isSubsBy}(Z\_st, \text{Starship\_Label})$ , meaning that only individuals that have property  $\text{hasType}$  equal to  $\text{Starship\_Label}$  can substitute for  $Z\_st$ .

Table 5. Zone\_MFrag Nodes in MEBN and PR-OWL

MEBN MFrag	PR-OWL Representation
$\text{IsA}(\text{TimeStep}, t_{\text{prev}})$	Implicit in the type declaration
$\text{IsA}(\text{Zone}, z)$	
$\text{IsA}(\text{Starship}, st)$	
$\text{IsA}(\text{TimeStep}, t)$	
$t_{\text{prev}} = \text{Prev}(t)$	$Z\_T_{\text{prev}}\text{PrevT\_context}$
	$Z\_T_{\text{prev}}\text{PrevT\_inner\_prevT}$
$z = \text{StarshipZone}(st)$	$Z\_ZS\text{ZoneST\_context}$
	$Z\_ZS\text{ZoneST\_inner\_SZoneST}$
$\text{CloakMode}(st)$	$Z\_Cloak\text{Mode\_input}$
$\text{ZoneMD}(z, t_{\text{prev}})$	$Z\_Zone\text{MD\_input}$
$t = !T0$	$Z\_T\text{equal}T0\_input$
$\text{ZoneMD}(z, t)$	$Z\_Zone\text{MD}$
$\text{ZoneNature}(z)$	$Z\_Zone\text{Nature}$
$\text{ZoneFShips}(z)$	$Z\_Zone\text{FShips}$
$\text{ZoneEShips}(z)$	$Z\_Zone\text{EShips}$

Therefore, even though there is no explicit reference to the “IsA” context nodes from Zone MFrag in the individuals displayed in Figure 31, the object property  $\text{hasOVariable}$  [3] linking the Zone\_MFrag with its respective ordinary variables implicitly conveys that subtyping restriction.

As an example of the mapping between MEBN and PR-OWL depicted in Table 5, representing the context node “ $z = \text{StarshipZone}(st)$ ” requires decomposing it into random

variable terms “equal( $z$ , StarshipZone( $st$ ))” and “StarshipZone( $st$ )”. In the PR-OWL ontology these RV terms are respectively represented by `Z_ZSZoneST_context` and `Z_ZSZoneST_inner_SZoneST`, both individuals of class `Context`.

The other properties depicted in Figure 31 are the object property `hasSkolem` [4], which links a quantifier `MFrag` with its respective Skolem constants, and the properties `hasResidentNode` [5], `hasInputNode` [6], and `hasContextNode` [7], all subproperties of `hasNode`.

Figure 32 portrays the representation of node `ZoneMD( $z$ ,  $t$ )`, from the `ZoneMFrag`. Object property `isNodeFrom` [1] provides the link between the node and its `MFrag`. Further structural information is provided by the parent list formed with the object property `hasParent` [2], the object property `isResidentNodeIn` [6] and the properties that link `ZoneMD( $z$ ,  $t$ )` with its “copies” (instances in which the node is used as input and/or context node in other `MFrag`s), which are `hasContextInstanceIn` [7] and `hasInputInstanceIn` (not visible in the picture).

`ZoneMD( $z$ ,  $t$ )` is a resident node, so it has a probability distribution conditioned on its parents. The link between an individual of class `Domain_res` and the many possible representations of its probability distribution is provided by the object property `hasProbDist` [3]. Subsection 5.2.2 explains the different possibilities of representing probability distributions in PR-OWL.

The list of possible states of `ZoneMD( $z$ ,  $t$ )` is made using the object property `hasPossibleValues` [4], while its arguments (ordinary variables  $z$  and  $t$ ) are linked using the `hasArgument` object property [5]. Note that property `hasArgument` doesn’t actually

point to an individual of class *OVariable*, which is the class that has the ordinary variables  $z$  and  $t$  represented (as  $Z\_z$  and  $Z\_t$  respectively). Instead, it points to individuals of class *SimpleArgRelationship*, a subclass of *ArgRelationship*. These two classes are reified relations specifying the many possibilities of arguments in a random variable. In this specific case (i.e. the  $\text{ZoneMD}(z, t)$  node), the two arguments are *OVariables*, so links to both are represented by individuals of the class *SimpleArgRelationship*, which works as a pointer to individuals of Class *OVariable* only. When a node has composite arguments, the parent class *ArgRelationship* should be used, since it works as a pointer to individuals of classes *OVariable*, *Node*, *Entity*, and *Skolem*.

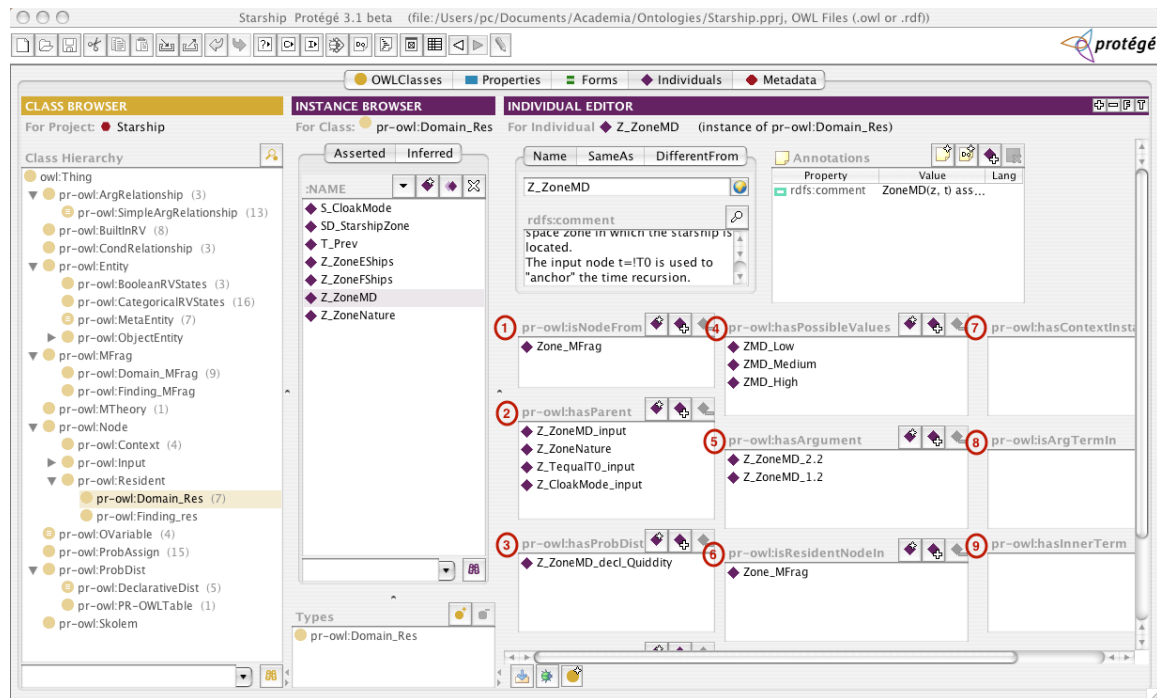


Figure 32. ZoneMD Resident Node

Object properties `isArgTermIn` [8] and `hasInnerTerm` [9] provide further support to reified relations, by keeping track of the complex relationships in which each node is

participating. The use of reification is also important for representing probability distributions in PR-OWL, which may be conveyed in different ways.

### 5.2.2 Representing a Probability Distribution

Representing probability distributions is a key issue in achieving a balance between interoperability and conciseness. Proprietary formats usually convey all the necessary information in a compact way, thus simply using that format in a `xsd:string` to convey that information is an attractive option. However, this option ties the ontology to a specific format that might not be universally known or might be inappropriate to a range of applications. Also, annotating probability distributions might reduce the ability to use that data in complex environments with many systems working with different formats, rules or requirements.

PR-OWL is supposed to facilitate interoperability and thus should be as flexible as possible in terms of how to represent probability distributions. Therefore, it allows using multiple declarative distributions and/or a RDF table format to represent the probability distribution of a given RV.

Each probability distribution can be expressed in different formats using PR-OWL's declarative distributions represented via the `DeclarativeDist` class, which is depicted in Figure 33. Possible formats include Netica tables, Netica equations, Quiddity formulas, MEBN syntax, and others. However, the declaration itself is stored as a string so parsers should be compatible with the specific text format of each declaration.

Every individual of class `DeclarativeDist` has an object property `isProbDistOf` [1] linking it with their respective resident node. A datatype property `isRepresentedAs` [2]

defines how a given declarative probability distribution is expressed. A datatype property `isDefault` [3] flags it (or not) as a default distribution. finally, a datatype `hasDeclaration` [4] includes the probability distribution itself in the format previously defined.

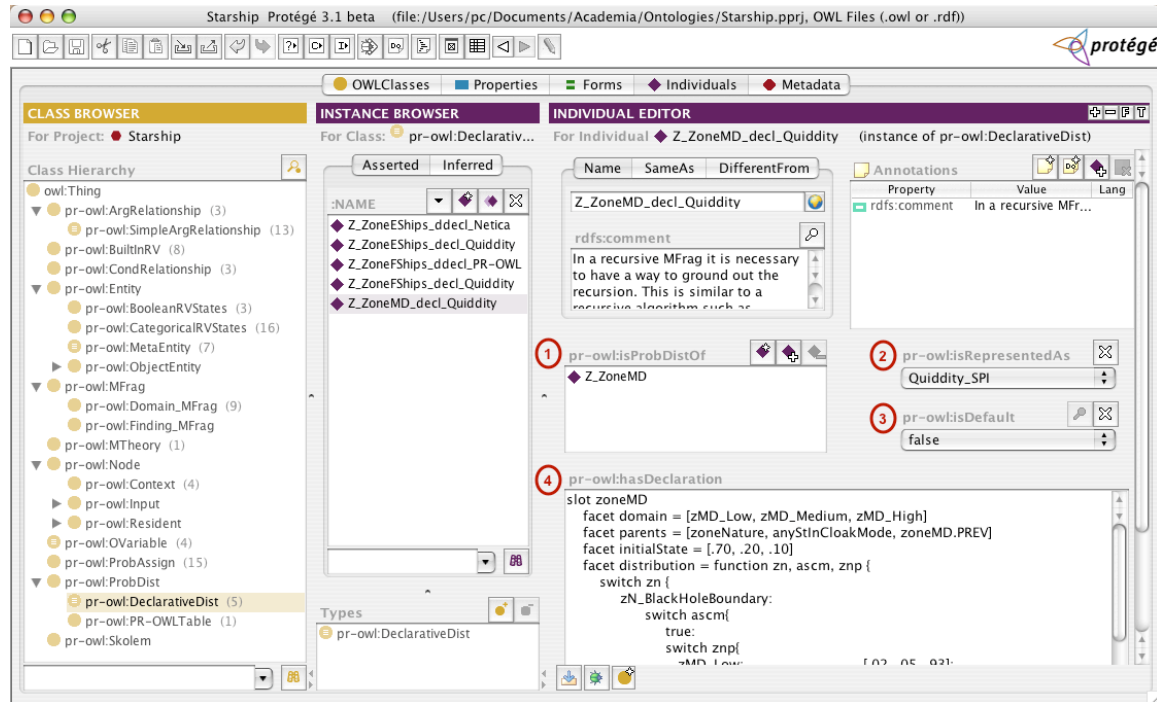


Figure 33. Declarative Distributions in PR-OWL

PR-OWL tables have a different representational scheme. Each individual of class `PR-OWLTable` is actually a label that links the many components that collectively form a probability distribution of a resident node. As an example, the individual `Z_ZoneFShips_table` has three properties: `isDefault`, which states whether or not that individual represents a default probability distribution, `isProbDistOf`, which links the individual with the node it represents (`Z_ZoneFShips` in this case), and `hasProbAssign`, which links the individual with all the individuals of class `ProbAssign` that collectively form the probability distribution of node `Z_ZoneFShips`. One of those `ProbAssign` individuals is `Z_ZoneFShips_table_2.3`, which is depicted in Figure 34.

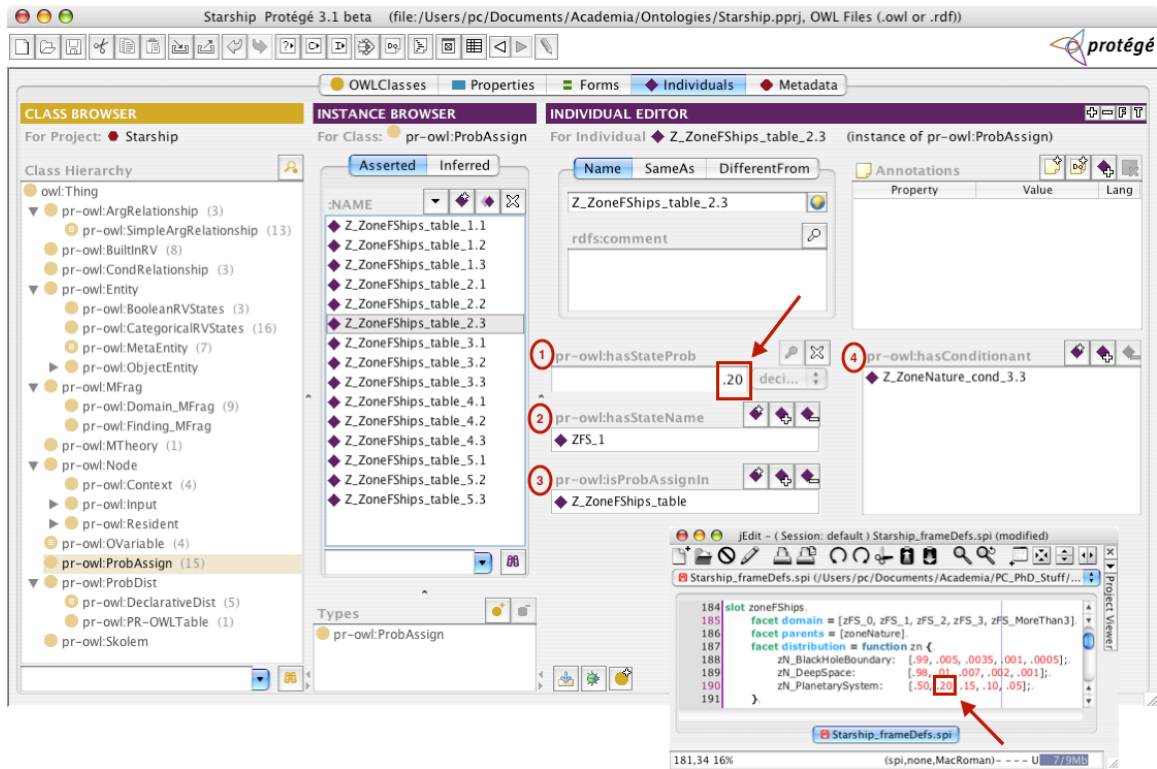


Figure 34. A Probabilistic Assignment in a PR-OWL Table

`Z_ZoneFShips_table_2.3` corresponds to the probability assigned to the second state of node `ZoneFShips` (`ZFS_1`) given that its parent node has value `ZN_PlanetarySystem` (the third state of that parent). The probability itself (.20) is represented as a `xsd:decimal` that is linked to `Z_ZoneFShips_table_2.3` via the datatype property `hasStateProb` [1]. The link between the `ProbAssign` individual and the state of `ZoneFShips` it refers to is made via the object property `hasStateName` [2], while property `isProbAssignIn` [3] links the probability assignment to the table it belongs. Finally, each probability assigned to a state of a variable is conditioned to a combination of states of the parents of that variable. Object property `hasConditionant` [4] links a `ProbAssign` individual to the individuals of class `CondRelationship` that collectively form such a combination of parents. `CondRelationship` is a reified relation linking a parent with one

of its possible states, and a set of CondRelationship individuals represents the combination of parents' states to which a given probability assignment is conditioned.

ZoneFShips has only one parent, so there is only one conditionant listed (Z\_ZoneNature\_cond\_3.3), which is an individual of the reified class CondRelationship that links node ZoneNature with its third state (ZN\_PlanetarySystem). If ZoneFShips had four parents, then four individuals of class CondRelationship (i.e. one for each parent) would have to be listed in order to represent the combination of parents under which that probability assignment is valid.

One issue regarding the probability assignment is the use of xsd:decimal to convey a probability value, when the ideal situation would be to use a datatype that specifically covers the numerical range of probabilities (i.e. 0 to 1, including both extremities). However, at the time of this writing, OWL has no support for user-defined datatypes, so the closest datatype allowed by OWL is xsd:decimal.

Although applications or plugins should be written to prevent invalid entries for probabilities, relying on external plugins to enforce this requirement is not an acceptable option. Therefore, a more robust solution must be sought. In the case of a future consideration of PR-OWL as a basis for a W3C Recommendation for representing uncertainty in the Semantic Web, a special datatype covering the numerical range of probabilities must be included. A very suitable name for such datatype is “prob” (pr-owl:prob), which has already been proposed by other researchers in this field (e.g., Ding & Peng, 2004).



PR-OWL tables represent probabilities in a format that is highly interoperable, since each cell contains links to all the elements that are necessary for specifying the conditions in which the probability inside that cell applies. Also, those elements are available in a non-proprietary, syntax-independent format, which makes it easier to be retrieved by diverse applications without the need for a format conversion. Yet, building PR-OWL tables the way it was done in this work is not a feasible option for a real life application or plugin. Fortunately, all the above steps can be avoided by developing automated tools. The next Section briefly covers such possibilities.

### 5.3 A Proposed Operational Concept for Implementing PR-OWL

In its current stage, PR-OWL contains only the basic representation elements that provide a means of representing any MEBN-based model. Such a representation could be used by a Bayesian tool (acting as a probabilistic ontology reasoner) to perform inferences to answer queries and/or to learn from newly incoming evidence via Bayesian learning.

However, building MFragments and all their elements in a probabilistic ontology is a manual, error prone, and tedious process. Avoiding errors or inconsistencies requires very deep knowledge of the logic and of the data structure of PR-OWL. Without considering the future paths to be followed by research on PR-OWL (i.e. whether it will be kept as an upper ontology or transformed into an actual extension to the OWL language), the framework provided in this Dissertation makes it already possible to facilitate probabilistic ontology usage and editing by developing plugins to current OWL editors.

Figure 35 illustrates an example of such a concept. In that figure, a possible plugin for the OWL Protégé editor (which is itself an OWL plugin) shows a graphical construction of an MFrag being performed in a very similar fashion as a BN is constructed in a graphical package such as Netica™.

In this proposed scheme, in order to build an MFrag a user would only have to select the icon of the node he/she wants to create (e.g. resident, input, context, etc.), connect that node with its parents and children, and enter its basic characteristics (i.e. name, probability distribution, etc.) either by double-clicking on it or via another GUI-related facility.

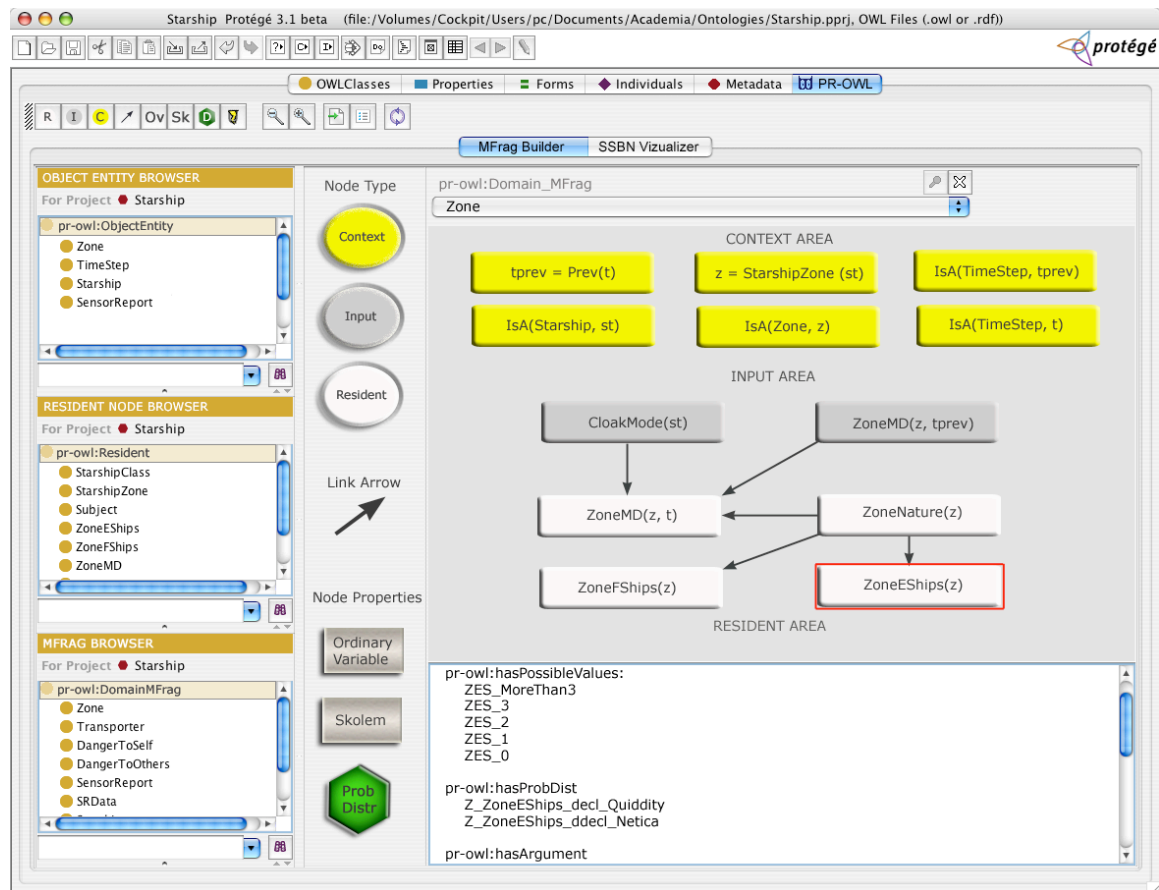


Figure 35. Snapshot of a Graphical PR-OWL Plugin

The idea of such a plugin is to hide from users the complex constructs required to convey the many details of a probabilistic ontology, such as the reified relationships, composite RV term constructions (with or without quantifiers and Skolem constants), and others. In the figure, the Zone MFrag was selected from the combo box in the top of the viewing area, thus information about its nodes is displayed in a graphical format that allows the user to build more nodes, edit or view the existing ones. and then chose node ZoneEShips(z) so it appears highlighted (a red box around it) and all its data is shown in the lower square.

Tedious tasks such as building a PR-OWL table with many cells could be carried out much more quickly and with fewer errors, thus providing a boost in productivity. In the probability table case, the user would only have to fill the probabilities in the correct cells of a CPT's graphical display and the plugin would build their respective PR\_OWL constructs.

Another point of usage improvement is the intrinsic syntax check provided by a guided construction. As an example, when writing a composite RV term, the user would not have to actually write the complex reified relations (ArgRelationships, Skolem constants, OVariables, Inner terms, etc). Instead, a menu with the allowed connectives would be available so his/her task would be reduced to enter the arguments of the formula and embed the connectives the way he/she wants. The final result would be a valid formula that would then be transformed in PR-OWL syntax by the plugin.

This brief idea of an operational concept barely scratches the surface of the many possibilities for the technology presented here, and its purpose is to point out one such

possibility. As previously stated, the present dissertation is focused on defining a coherent, comprehensive probabilistic framework for the Semantic Web, in a way that any probabilistic system could be represented and made available to perform tasks such as plausible inference and Bayesian learning. Therefore, implementing a plugin such as the one envisioned here is a development task that is outside the scope of this dissertation research. Nonetheless, it takes an important first step toward making probabilistic ontologies a reality. By opening the door to wide use of PR-OWL probabilistic ontologies, the present research makes a significant contribution to realizing the Semantic Web vision.

## Chapter 6 Conclusion and Future Work

### 6.1 Summary of Contributions

The main objective of this research effort was to establish a framework that enables the use of Bayesian theory for representing and reasoning under uncertainty in the context of the Semantic Web. The key step for achieving such objective was the introduction of probabilistic ontologies, which were formally defined in Chapter 5.

In order to provide the initial conditions for the future spread of probabilistic ontologies, we have developed a complete, modularized set of new definitions for the OWL language, which collectively form a coherent framework for building ontologies that are able to represent uncertainty from concepts of a given domain with full probabilistic first-order logic expressiveness.

Probabilistic ontologies written under this framework achieve a principled representation of uncertainty and allow for the use of different probabilistic reasoning systems as a means to perform plausible reasoning and learning from data on the MTheories represented in PR-OWL format.

The contributions of this research effort also included the development and formalization of a typed version of MEBN logic. This extended version was needed as a means to achieve full compatibility with current Semantic Web languages, including OWL.

A full implementation of MEBN logic and its typed extension does not yet exist. However, Quiddity\*Suite is a powerful Bayesian probabilistic reasoning system that is capable of being applied as a PR-OWL reasoner. Therefore, we have also developed a set of rules for translating an MTheory written using the typed version of MEBN into a probabilistic model in IET's Quiddity\*Suite format.

These rules were applied to the Starship MTheory specially developed for this research, and resulted in a running Quiddity\*Suite model. The Starship MTheory includes some of the most complex aspects that can be expressed with MEBN logic, such as recursions, nodes with many uncertain parents, context constraints expressed as first-order logic sentences with and without quantifiers, etc. Therefore, having achieved a Quiddity\*Suite model capable of building any SSBN based on the original MTheory is a valid proof of concept of the feasibility of using Quiddity\*Suite as a PR-OWL reasoner. The source code for the Quiddity Starship model is provided in the Appendix A of this dissertation.

In order to demonstrate the feasibility of representing a complex MTheory using the concepts laid out in Chapter 5, the very same case study was used as a basis for writing a probabilistic ontology containing all the elements from the original model and exploring different possibilities for representing a probability distribution. The resulting PR-OWL ontology is logically equivalent to the original generative MTheory, and thus can be utilized as the basis for generating SSBNs to answer queries posed to the model. In addition, the representation of Finding MFragments was also covered, as a means to

demonstrate how PR-OWL ontologies can incorporate new information, either via user insertion or by means of Bayesian learning from data.

Therefore, the upper ontology presented here is capable of representing any MTheory, including both generative and MTheories with findings. In addition, it allows users to define probabilistic ontologies using a RDF-based syntax that is compatible with current OWL ontologies. Furthermore, translators could be written for third-party, of-the-shelf probabilistic reasoners to make use of the ontology to perform Bayesian inference and learning. These capabilities were demonstrated by creating the case study ontology, translating its definitions into Quiddity\*Suite and performing probabilistic inferences over it, a process that is documented in the appendices.

## 6.2 A Long Road with Bright Signs Ahead

The proposed framework can be understood as an initial solution situated in a middle ground between the extension approaches employed in OWL-S and SRWL. In common with the first is the fact that no actual extension to OWL semantics and abstract syntax is performed at this time, since it is also an OWL upper ontology. Similarly to the latter, PR-OWL also has the need for specialized tools in order to realize its full potential, while also including concepts (e.g. the prob datatype, FOL connectives, quantifiers, etc.) that could greatly expand OWL expressiveness if adopted as a standard.

Even though it is possible to represent a complex probabilistic system using PR-OWL definitions, performing plausible reasoning and learning from data requires an external tool (e.g. Quiddity\*Suite). It is true that some preliminary consistency check and

other OWL-DL features are possible using PR-OWL (which is OWL-DL compliant), and that any complex system can still be written in PR-OWL and be interpreted using different probabilistic reasoning systems, provided that PR-OWL plugins are written for capturing the data inside probabilistic ontologies in each package's native format.

Apart from the need for developing plugins for probabilistic packages so they can be used as plausible reasoners, a specific PR-OWL plugin for current OWL ontology editors remains a priority for future efforts. The process used here for writing probabilistic ontologies can be greatly improved via automation of most of the steps in the ontology building, mainly in the part of writing composite RV terms, but also for consistency checking, reified relations and other tasks that demand unnecessary awareness of the inner workings of the present solution. Once implemented, such plugin has the potential to make probabilistic ontologies a natural, powerful tool for helping to realize the Semantic Web vision.

Furthermore, the technology has the potential to be used in important applications outside the Semantic Web, as we discuss in Appendix C. In that discussion, our main point is that the proper use of probability information can help to establish reliable, more general semantic mapping schemas by means of probabilistic ontologies, which can then be applied in applications spanning diverse domains, since it relies on a meta-ontology (i.e. a ontology about ontologies), carrying no domain information, which has the mappings between two or more ontologies as its instances.

That is, PR-OWL has the potential for application in other semantic mapping solutions such as the DTB case study presented in Section C.1 of the appendices. It could



also be applied to facilitate interoperability between systems as discussed in the Wise Pilot cases study, presented in Section C.2. The present work thus represents a step toward a general-purpose solution for the semantic mapping problem.

Finally, the most important requirement for adoption of a language is the standardization process. This process goes significantly beyond academic research and thus falls outside the scope of the present work. Nonetheless, we are confident of its feasibility, which we believe having demonstrated in this effort, and of its desirability, given its potential to help solve many of the obstacles that stand in the way of realizing the W3C's vision for the Semantic Web.

## Bibliography

## Bibliography

- Adams, J. B. (1976). A Probability Model for Medical Reasoning and the MYCIN Model. *Mathematical Biosciences*, 32(1/2), 177-186.
- Alberts, D. S., Garstka, J. J., & Stein, F. P. (1999). *Network Centric Warfare: Developing And Leveraging Information Superiority*. Washington, D.C., USA: National Defense University Press.
- Alghamdi, G., Laskey, K. B., Wang, X., Barbara, D., Shackleford, T., Wright, E. J., et al. (2004). Detecting Threatening Behavior Using Bayesian Networks, Conference on Behavioral Representation in Modeling and Simulation - BRIMS. Arlington, VA, USA.
- Alghamdi, G., Laskey, K. B., Wright, E. J., Barbará, D., & Chang, K.-C. (2005). Modeling insider user behavior using Multi-Entity Bayesian Networks. In *Proceedings of the Tenth International Command and Control Research Technology Symposium (10<sup>th</sup> ICCRTS)*. McLean, VA, USA: CCRP/DOD publications.
- Arpírez, J. C., Corcho, O., Fernández-López, M., & Gómez-Pérez, A. (2001). WebODE: A Scalable Workbench for Ontological Engineering, *International Conference on Knowledge Capture* (pp. 6-13). Victoria, British Columbia, Canada: ACM Press.
- Baader, F.; & Nutt, W. (2003). Basic Description Logics. Chapter in *The Description Logics Handbook: Theory, Implementation and Applications*. Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; Patel-Schneider, P.; editors. 1<sup>st</sup> edition, chapter 2, pages 47-100. Cambridge, UK: Cambridge University Press.
- Ball, W. W. R. (2003). *A Short Account of the History of Mathematics*. New York, NY, USA: Main Street Books (originally published in 1908).

- Bangsø, O., & Wuillemin, P.-H. (2000). Object Oriented Bayesian Networks: A Framework for Topdown Specification of Large Bayesian Networks and Repetitive Structures. Technical Report No. CIT-87.2-00-obphw1. Department of Computer Science, Aalborg University, Aalborg, Denmark.
- Baum, L. E., & Petrie, T. (1966). Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *Annals of Mathematical Statistics*, 37, 1554-1563.
- Berners-Lee, T., & Fischetti, M. (2000). *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. 1<sup>st</sup> edition. New York, NY, USA: HarperCollins Publishers.
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web, Scientific American Digital (pp. 7): Scientific American, Inc.
- Bobrow, D. G., & Winograd, T. (1977). An Overview of KRL, a Knowledge Representation Language. *Cognitive Science*, 1(1), 3-46.
- Boley, H., & Tabet, S. (2004). The Rule Markup Initiative. Retrieved May 29, 2005, from <http://www.ruleml.org/>
- Booker, L. B., & Hota, N. (1986, August 8-10). Probabilistic Reasoning about Ship Images. Paper presented at the Second Annual Conference on Uncertainty in Artificial Intelligence, University of Pennsylvania, Philadelphia, PA.
- Bournez, C., & Hawke, S. (2004, November 23). Team Comment on the OWL-S Submission. Retrieved June 29, 2005, from <http://www.w3.org/Submission/2004/07/Comment>
- Brachman, R. J. (1977). What's in a Concept: Structural Foundations for Semantic Networks. *International Journal of Man-Machine Studies*, 9(2), 127-152.
- Brachman, R. J., & Schmolze, J. G. (1985). An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2), 171-216.

- Buchanan, B. G., & Shortliffe, E. H. (1984). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA, USA: Addison-Wesley.
- Buntine, W. L. (1994a). Learning with Graphical Models. Technical Report No. FIA-94-03. NASA Ames Research Center, Artificial Intelligence Research Branch.
- Buntine, W. L. (1994b). Operations for Learning with Graphical Models. *Journal of Artificial Intelligence Research*, 2, 159-225.
- Calvanese, D.; & De Giacomo, G. (2003). Expressive Description Logics. Chapter in *The Description Logics Handbook: Theory, Implementation and Applications*. Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; Patel-Schneider, P.; editors. 1<sup>st</sup> edition, chapter 5, pages 184-225. Cambridge, UK: Cambridge University Press.
- Cardelli, L., & Wegner, P. (1985). On Understanding Types, Data Abstraction and Polymorphism. *Computing Surveys*, 17(4), 471-522.
- Carnap, R. (1950). *Logical Foundations of Probability*. Chicago, IL, USA: University of Chicago Press.
- Charniak, E. (1991). Bayesian Networks without Tears. *AI Magazine*, 12, 50-63.
- Charniak, E., & Goldman, R. P. (1989a). Plan Recognition in Stories and in Life. Paper presented at the Fifth Workshop on Uncertainty in Artificial Intelligence, Mountain View, California.
- Charniak, E., & Goldman, R. P. (1989b). A Semantics for Probabilistic Quantifier-Free First-Order Languages with Particular Application to Story Understanding. Paper presented at the Eleventh International Joint Conference on Artificial Intelligence, August 1989, Detroit, Michigan, USA.
- Cheng, J., & Druzdzel, M. J. (2000). AIS-BN: An Adaptive Importance Sampling Algorithm for Evidential Reasoning in Large Bayesian Networks. *Journal of Artificial Intelligence Research*, 13, 155-188.

- Clottes, J. E., Chauvet, J. M., Brunel-Deschamps, E., Hillaire, C., Saugas, J. P., Evin, J., et al. (Eds.). (1995). *Les Peintures Paléolithiques de la Grotte Chauvet Pont-d'Arc à Vallon Pont-d'Arc (Ardèche, France): Datations directes et indirectes par la méthode du radiocarbone* (Vol. 320). Paris, France: l'Académie de Sciences.
- CNN.com. (1998). Rationalizing Treason: An Interview with Aldrich Ames. Cold War Experience - Espionage Series Retrieved January 20, 2005, from <http://www.cnn.com/SPECIALS/cold.war/experience/spies/interviews/ames>.
- CNN.com. (2001). The Case Against Robert Hanssen: An FBI Insider and Admitted Spy. In-Depth Special Series Retrieved January 20, 2005, from <http://www.cnn.com/SPECIALS/2001/hanssen/>
- Codd, E. F. (1970). A Relational Model for Large Shared Data Banks. *Communications of the ACM*, 13(6), 377-387.
- Cohen, L. J. (1989). *An Introduction to the Philosophy of Induction and Probability*. Oxford, UK: Clarenton Press.
- Connolly, D., Khare, R., & Rifkin, A. (1997). The Evolution of Web Documents: The Ascent of XML. *World Wide Web Journal* (special issue on XML), 2(4), 119-128.
- Cooper, G. F. (1987). Probabilistic Inference using Belief Networks is Np-Hard. Paper No. SMI-87-0195. Knowledge Systems Laboratory, Stanford University. Stanford, CA, USA.
- Cooper, G. F., & Herskovits, E. (1992). A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 9, 309-347.
- Costa, P. C. G. (1999). The Fighter Aircraft's Autodefense Management Problem: A Dynamic Decision Network Approach. Master of Science Thesis, School of Information Technology and Engineering, George Mason University. Fairfax, VA, USA.
- Costa, P. C. G., Laskey, K. B., Takikawa, M., Pool, M., Fung, F., & Wright, E. J. (2005). MEBN Logic: A Key Enabler for Network Centric Warfare. In *Proceedings of the*

Tenth International Command and Control Research and Technology Symposium (10<sup>th</sup> ICCRTS). Mclean, VA, USA: CCRP/DOD publications.

Cox, R. T. (1946). Probability, Frequency and Reasonable Expectation. *American Journal of Physics*, 14, 1-13.

Cristianini, N., Shawe-Taylor, J., & Lodhi, H. (2001). Latent Semantic Kernels. Paper presented at the 18th International Conference on Machine Learning, San Francisco, CA, USA.

DaConta, M. C., Obrst, L. J., & Smith, K. T. (2003). *The Sematic Web: A Guide to the Future of Xml, Web Services, and Knowledge Management*. Indianapolis, IN, USA: Wiley Publishing, Inc.

Dagum, P., & Luby, M. (1993). Approximating Probabilistic Inference in Bayesian Belief Networks is Np-Hard. *Artificial Intelligence*, 60, 141-153.

Damas, L., & Milner, R. (1982). Principal Type Schemes for Functional Programs. Paper presented at the 9th ACM Symposium on Principles of Programming Languages. Albuquerque, NM, USA.

de Finetti, B. (1990). *Theory of Probability: A Critical Introductory Treatment*. New York, NY, USA: John Wiley & Sons. Originally published in 1974.

Dempster, A. P. (1967). Upper and Lower Probabilities Induced by a Multivalued Mapping. *Annals of Mathematical Statistics*, 38, 325-339.

Ding, Z., & Peng, Y. (2004). A Probabilistic Extension to Ontology Language OWL. Paper presented at the 37<sup>th</sup> Annual Hawaii International Conference on System Sciences (HICSS'04). Jan, 5-8, 2004. Big Island, Hawaii, USA.

Druzdzal, M. J., & Simon, H., A. (1993). Causality in Bayesian Belief Networks. Paper presented at the Ninth Annual Conference on Uncertainty in Artificial Intelligence (UAI-93). San Francisco, CA, USA.

- Druzdzal, M. J., & Yuan, C. (2003). An Importance Sampling Algorithm Based on Evidence Pre-Propagation. Paper presented at the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence. Acapulco, Mexico.
- Elliott, R. J., Aggoun, L., & Moore, J. B. (1995). *Hidden Markov Models: Estimation and Control*. New York, NY, USA: Springer-Verlag.
- Fellbaum, C. (Ed.). (1998). *Wordnet, An Electronic Lexical Database*. Cambridge, MA, USA: The MIT Press.
- Fikes, R. E., & Kehler, T. P. (1985). The Role Of Frame-Based Representation in Knowledge Representation and Reasoning. *Communications of the ACM*, 28(9), 904-920.
- Fine, T. L. (1973). *Theories of Probability: An Examination of Foundations*. New York, NY, USA: Academic Press.
- Frege, G. (1879) *Begriffsschrift, Eine Der Arithmetischen Nachgebildete Formelsprache Des Reinen Denkens*, Halle a. S.: Louis Nebert. Translated as *Concept Script, A Formal Language of Pure Thought Modeled Upon that of Arithmetic*, by S. Bauer-Mengelberg in J. vanHeijenoort (ed.), *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*. Cambridge, MA, USA: Harvard University Press, 1967.
- Friedman, N., & Koller, D. (2000). Being Bayesian about Network Structure. Paper presented at the Sixteenth Conference on Uncertainty in Artificial Intelligence. San Mateo, California, USA.
- Fukushige, Y. (2004). Representing Probabilistic Knowledge in the Semantic Web, W3C Workshop on Semantic Web for Life Sciences. Cambridge, MA, USA.
- Fung, F., Laskey, K. B., Pool, M., Takikawa, M., & Wright, E. J. (2004). PLASMA: Combining Predicate Logic and Probability for Information Fusion and Decision Support. Paper presented at the AAAI Spring Symposium. Stanford, CA, USA.
- Fung, R., & Chang, K. C. (1989). Weighing and Integrating Evidence for Stochastic Simulation in Bayesian Networks. In M. Henrion, R. D. Shachter, L. N. Kanal &



- J. F. Lemmer (Eds.), *Uncertainty in Artificial Intelligence 5* (pp. 209-219). New York, NY, USA: Elsevier Science Publishing Company, Inc.
- Fung, R., & del Favero, B. (1994). Backward Simulation in Bayesian Networks. Paper presented at the Tenth Annual Conference on Uncertainty in Artificial Intelligence. San Francisco, CA, USA.
- Gelman, A. (2003). *Bayesian Data Analysis*. 2<sup>nd</sup> edition. London, UK: Chapman and Hall.
- Getoor, L., Friedman, N., Koller, D., & Pfeffer, A. (2001). *Learning Probabilistic Relational Models*. New York, NY, USA: Springer-Verlag.
- Getoor, L., Koller, D., Taskar, B., & Friedman, N. (2000). Learning Probabilistic Relational Models with Structural Uncertainty. Paper presented at the ICML-2000 Workshop on Attribute-Value and Relational Learning: Crossing the Boundaries. Stanford, CA, USA.
- Gil, Y., & Ratnakar, V. (2004). Markup Languages: Comparison and Examples. Retrieved January 16, 2005, from <http://trellis.semanticweb.org/expect/web/semanticweb/comparison.html>
- Gilks, W., Thomas, A., & Spiegelhalter, D. J. (1994). A Language and Program for Complex Bayesian Modeling. *The Statistician*, 43, 169-178.
- Giugno, R., & Lukasiewicz, T. (2002). P-SHOQ(D): A Probabilistic Extension of SHOQ(D) for Probabilistic Ontologies in the Semantic Web. Paper presented at the European Conference on Logics in Artificial Intelligence (JELIA 2002), Sep 23-26. Cosenza, Italy.
- Goldfarb, C. F. (1996). The Roots of SGML - A Personal Recollection. Retrieved September 30, 2004, from <http://www.sgmlsource.com/history/roots.htm>
- Greco, S., Leone, N., & Rullo, P. (1992). COMPLEX: An Object-Oriented Logic Programming System. *IEEE Transactions on Knowledge and Data Engineering*, 4(4), 344-359.

- Grenander, U. (1995). *Elements of Pattern Theory*. Baltimore, MD, USA: Johns Hopkins University Press.
- Gruber, T. R. (1993). A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2), 199-220.
- Gu, T., Keng, P. H., & Qing, Z. D. (2004). A Bayesian Approach for Dealing With Uncertainty Contexts. Paper presented at the Second International Conference on Pervasive Computing, Apr 18-23. Vienna, Austria.
- Haarslev, V., & Möller, R. (2001). RACER System Description. Paper presented at the International Joint Conference on Automated Reasoning (IJCAR'2001). Siena, Italy.
- Hacking, I. (1965). *The Logic of Statistical Inference*. Cambridge, MA, USA: Cambridge University Press.
- Hacking, I. (1975). *The Emergence of Probability: A Philosophical Study of Early Ideas about Probability, Induction, and Statistical Inference*. Cambridge, MA, USA: Cambridge University Press.
- Hansson, O., & Mayer, A. (1989). Heuristic Search as Evidential Reasoning. Paper presented at the Fifth Workshop on Uncertainty in Artificial Intelligence. Windsor, Ontario, Canada.
- Heckerman, D., Geiger, D., & Chickering, D. M. (1995a). Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning* (20), 197-243.
- Heckerman, D., Mamdani, A., & Wellman, M. P. (1995b). Real-World Applications of Bayesian Networks. *Communications of the ACM*, 38(3), 24-68.
- Heckerman, D., Meek, C., & Koller, D. (2004). Probabilistic Models for Relational Data. Technical Report MSR-TR-2004-30, Microsoft Corporation, March 2004. Redmond, WA, USA.

- Heflin, J. (2004, February 10). OWL Web Ontology Language - Use Cases And Requirements. Retrieved December 04, 2005, from <http://www.w3.org/TR/2004/REC-webont-req-20040210/>
- Heinsohn, J. (1994). Probabilistic Description Logics. Paper presented at the Tenth Conference on Uncertainty in Artificial Intelligence (UAI-94), Jul 29-31. Seattle, WA, USA.
- Helsper, E. M., & van der Gaag, L. C. (2001). Ontologies for Probabilistic Networks: A Case Study in Oesophageal Cancer. Paper presented at the Thirteenth Dutch-Belgian Artificial Intelligence Conference. Amsterdam, The Netherlands.
- Hendler, J. (2004). Frequently Asked Questions on W3C's Web Ontology Language (OWL). Retrieved January 14, 2005, from <http://www.w3.org/2003/08/owlfaq.html>.
- Henrion, M. (1988). Propagation of Uncertainty by Probabilistic Logic Sampling in Bayes Networks. In J. F. Lemmer & L. N. Kanal (Eds.), *Uncertainty in Artificial Intelligence 2* (pp. 149-163). New York, NY, USA: Elsevier Science Publishing Company, Inc.
- Hillmann, D. (2001). Using Dublin Core. Retrieved June 28, 2005, from <http://dublincore.org/documents/usageguide/>.
- Hofmann, T. (1999). Probabilistic Latent Semantic Indexing. Paper presented at the 22<sup>nd</sup> Annual ACM Conference on Research and Development in Information Retrieval, August 15-19. Berkeley, CA, USA.
- Horrocks, I. (2002). DAML+OIL: A Reasonable Web Ontology Language. Keynote talk at the WES/CAiSE Conference. Toronto, Canada.
- Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., & Dean, M. (2004). SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission. Retrieved May 29, 2005, from <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>.

Horrocks, I., & Sattler, U. (2001). Ontology Reasoning in the SHOQ(D) Description Logic. Paper presented at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI 2001), Aug 4-10. Seattle, WA, USA.

Hotho, A., Staab, S., & Stumme, G. (2002). Text Clustering Based on Background Knowledge. Technical Report No. 425. Institute of Applied Informatics and Formal Description Methods AIFB, University of Karlsruhe. Karlsruhe, Germany.

Hotho, A., Staab, S., & Stumme, G. (2003). Wordnet Improves Text Document Clustering, Semantic Web Workshop at SIGIR-2003, 26<sup>th</sup> Annual International ACM SIGIR Conference. Toronto, Canada.

Jacobs, I., editor (2003, June 18). World Wide Web Consortium Process Document. Retrieved March 03, 2005, from <http://www.w3.org/2003/06/Process-20030618/cover.html>

Jaeger, M. (1994). Probabilistic Reasoning in Terminological Logics. Paper presented at the Fourth International Conference on Principles of Knowledge Representation and Reasoning (KR94), May 24-27. Bonn, Germany.

Jaeger, M. (1997). Relational Bayesian Networks. Paper presented at the 13th Annual Conference on Uncertainty in Artificial Intelligence (UAI97), August 1-3, Providence, RI, USA.

Jensen, F. V. (1996). *An Introduction to Bayesian Networks*. New York, NY, USA: Springer-Verlag.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. New York, NY, USA: Springer-Verlag.

Joachims, T. (1998). Text Categorization with Support Vector Machines: Learning With Many Relevant Features. Paper presented at the Tenth European Conference on Machine Learning, April 21 – 24. Chemnitz, Germany.

Jordan, M. I., (Ed.). (1999). *Learning in Graphical Models*. Cambridge, MA, USA: MIT Press.

- Kadane, J. B., Schervish, M. J., & Seidenfeld, T. (1999). *Rethinking the Foundations of Statistics*. New York, NY, USA: Cambridge University Press.
- Kadane, J. B., & Schum, D. A. (1996). *A Probabilistic Analysis of the Sacco and Vanzetti Evidence*. New York, NY, USA: John Wiley & Sons.
- Kalyanpur, A., Sirin, E., Parsia, B., & Hendler, J. (2004). Hypermedia Inspired Ontology Engineering Environment: SWOOP. International Semantic Web Conference (ISWC2004). Hiroshima, Japan.
- Keynes, J. M. (2004). *A Treatise on Probability*. New York, NY, USA: Dover Publications. Originally published in 1921.
- Kifer, M., Lausen, G., & Wu, J. (1990). Logical Foundations of Object-Oriented and Frame-Based Languages. Technical Report TR-90-003. University of Mannheim. Mannheim, Baden-Württemberg, Germany.
- Knublauch, H., Ferguson, R. W., Noy, N. F., & Musen, M. A. (2004). The Protégé OWL plugin: An Open Development Environment for Semantic Web Applications. Third International Semantic Web Conference (ISWC2004). Hiroshima, Japan.
- Kokkelink, S., & Schwänzl, R. (2001). Expressing Qualified Dublin Core in RDF / XML. Retrieved June 28, 2005, from <http://dublincore.org/documents/dcq-rdf-xml/> .
- Koller, D., Levy, A. Y., & Pfeffer, A. (1997). P-CLASSIC: A Tractable Probabilistic Description Logic. Paper presented at the Fourteenth National Conference on Artificial Intelligence (AAAI-97), July 27-31. Providence, RI, USA.
- Koller, D., & Pfeffer, A. (1997). Object-Oriented Bayesian Networks. Paper presented at the Thirteenth Conference on Uncertainty in Artificial Intelligence (UAI-97). San Francisco, CA, USA.
- Kolmogorov, A. N. (1960). *Foundations of the Theory of Probability*. 2<sup>nd</sup> edition. New York, NY, USA: Chelsea Publishing Co. Originally published in 1933.

- Korb, K. B., & Nicholson, A. E. (2003). *Bayesian Artificial Intelligence*. Boca Raton, FL, USA: Chapman & Hall/CRC Press.
- Langseth, H., & Nielsen, T. (2003). Fusion of Domain Knowledge with Data for Structured Learning in Object-Oriented Domains. *Journal of Machine Learning Research*, Special Issue on the Fusion of Domain Knowledge with Data for Decision Support, vol. 4, pp. 339-368, July 2003.
- Laplace, P. S. (1996). *A Philosophical Essay and Probabilities*. New York, NY, USA: Dover Publications. Originally published in 1826.
- Laskey, K. B. (2005, March 15). First-Order Bayesian Logic. Draft paper, Department of Systems Engineering and Operation Research, School of Information Technology and Engineering, George Mason University. Retrieved July 3, 2005, from <http://ite.gmu.edu/~klaskey/publications.html>
- Laskey, K. B., & Lehner, P. E. (1994). Metareasoning and the Problem of Small Words. *IEEE Transactions on Systems, Man and Cybernetics*, 24(11), 1643-1652.
- Lauritzen, S., & Spiegelhalter, D. J. (1988). Local Computation and Probabilities on Graphical Structures and their Applications to Expert Systems. *Journal of Royal Statistical Society*, 50(2), 157-224.
- Lee, P. M. (2004). *Bayesian Statistics: An Introduction*. 3<sup>rd</sup> edition. London, UK: Edward Arnold Publishers.
- Lewis, D. (1980). A Subjectivist's Guide to Objective Chance. In *Studies in Inductive Logic and Probability*, Vol II. Berkeley, CA, USA: University of California Press.
- Littman, M. L., Letsche, T. A., Dumais, S. T., & Landauer, T. K. (1997, March 24-26). Automatic Cross-Linguistic Information Retrieval using Latent Semantic Indexing. Paper presented at the AAAI Spring Symposium on Cross-Language Text and Speech Retrieval, Stanford University. Stanford, CA, USA.
- Lucas, P. J. F., van der Gaag, L. C., & Abu-Hanna, A. (2001, July 1<sup>st</sup>). Workshop on Bayesian Models in Medicine. Paper presented at the European Conference on Artificial Intelligence in Medicine. Cascais, Portugal.

- Mahoney, S. M., & Laskey, K. B. (1998). Constructing Situation Specific Networks. In UAI'98: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, pages 370-378. July 24-26, University of Wisconsin Business School. Madison, WI, USA.
- Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., et al. (2004). OWL-S: Semantic Markup for Web Services. Retrieved June 29, 2005, from <http://www.daml.org/services/owl-s/1.1/overview/>.
- Miller, E., & Hendler, J. (2004). Web Ontology Language (OWL). Retrieved January 14, 2004, from <http://www.w3.org/2004/OWL/>.
- Miller, E., & Hendler, J. (2005, February 04). Web Ontology Language (OWL). Retrieved June 28, 2005, from <http://www.w3.org/2004/OWL/>.
- Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. (1990). Five Papers on WordNet. Technical Report No. 43, Cognitive Science Laboratory, Princeton University. Princeton, NJ, USA.
- Milner, R. (1978). A Theory of Type Polymorphism in Programming. *Journal of Computer and System Sciences*, 17, 348-375.
- Minsky, M. L. (1975). Framework for Representing Knowledge. In *The Psychology of Computer Vision*. P. H. Winston (Eds.), pages 211-277. New York, NY: McGraw-Hill.
- Mitchell, J. C. (1984). Type Inference and Type Containment. Paper presented at the International Symposium on Semantics of Data Types, June 27-29. Sophia-Antipolis, France.
- Murphy, K. (1998). Dynamic Bayesian Networks: Representation, Inference and Learning. Doctoral Dissertation, University of California. Berkeley, CA, USA.
- Neapolitan, R. E. (1990). *Probabilistic Reasoning in Expert Systems: Theory and Algorithms*. New York, NY, USA: John Wiley and Sons, Inc.

Neapolitan, R. E. (2003). *Learning Bayesian Networks*. New York, NY, USA: Prentice Hall.

Noy, N. F., Fergerson, R. W., & Musen, M. A. (2000). The Knowledge Model of Protégé-2000: Combining Interoperability and Flexibility. Paper presented at the Twelfth International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000). Juan-les-Pins, France.

Noy, N. F., & Rector, A. (2004). Defining N-ary Relations on the Semantic Web: Use with Individuals. W3C Working Draft Retrieved May 06, 2005, from <http://www.w3.org/TR/2004/WD-swbp-n-aryRelations-20040721/>.

Noy, N. F., Sintek, M., Decker, S., Crubezy, M., Fergerson, R. W., & Musen, M. A. (2001). Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 16(2), 60-71.

Oliver, R. M., & Smith, J. Q. (1990). *Influence Diagrams, Belief Nets and Decision Analysis*. 1<sup>st</sup> edition. New York, NY, USA: John Willey & Sons Inc.

Page, L. B. (1988). *Probability for Engineering with Applications to Reliability*. New York, NY, USA: Computer Science Press, Inc.

Patel-Schneider, P. F. (2005, April 11). A Proposal for a SWRL Extension Towards First-Order Logic. W3C Member Submission. Retrieved May 29, 2005, from <http://www.w3.org/Submission/2005/SUBM-SWRL-FOL-20050411/>.

Patel-Schneider, P. F., Hayes, P., & Horrocks, I. (2004, February 10). OWL Web Ontology Language - Semantics and Abstract Syntax. W3C Recommendation. Retrieved December 05, 2004, from <http://www.w3.org/TR/owl-semantics/>.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA, USA: Morgan Kaufmann Publishers.

Pearl, J. (2000). *Causality: Models, Reasoning, and Inference*. Cambridge, U.K.: Cambridge University Press.



- Peirce, C. S. (1885). On the Algebra of Logic. *American Journal of Mathematics*, 7, 180-202.
- Pfeffer, A. (2001). IBAL: A Probabilistic Rational Programming Language International. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001)*, August 4-10, vol. 1, pp. 733-740. Seattle, WA, USA.
- Pfeffer, A., Koller, D., Milch, B., & Takusagawa, K. T. (1999). SPOOK: A System for Probabilistic Object-Oriented Knowledge Representation. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pp. 541-550, July 30 – August 1. Stockholm, Sweden.
- Polleres, A.; Bussler, C.; Travers, G.; Domingue, J. B.; & Burdett, D. (2005, April 4). Web Service Modeling Ontology (WSMO) Submission. W3C Member Submission. Retrieved June 29, 2005, from <http://www.w3.org/Submission/2005/06/>.
- Pool, M., & Aikin, J. (2004). KEEPER and Protégé: An Elicitation Environment for Bayesian Inference Tools. Paper presented at the Workshop on Protégé and Reasoning held at the Seventh International Protégé Conference, July 6 – 9. Bethesda, MD, USA.
- Pool, M. (2004). An OWL Based Implementation of Quiddity\*Modeler. Technical Report. Information Extraction and Transport, Inc., July 2004. Rosslyn, VA, USA.
- Popper, K. R. (1957). The Propensity Interpretation of the Calculus of Probability and the Quantum Theory. In Stefan Körner, ed. *Observation and Interpretation: A Symposium of Philosophers and Physicists*. Proceedings of the Ninth Symposium of the Colston Research Society held in the University of Bristol, April 1st–April 4th, 1957. London: Butterworth Scientific Publications, 1957, 65–70.
- Popper, K. R. (1959). The Propensity Interpretation of Probability. *British Journal for the Philosophy of Science* 10 (1959), 25–42.
- Porter, M. F. (1980). An Algorithm for Suffix Stripping. *Program - Electronic Library and Information Systems*, 14(3), p.130-137.

- Powers, S. (2003). *Practical RDF*. 1<sup>st</sup> edition. Sebastopol, CA, USA: O'Reilly & Associates, Inc.
- Press, S. J. (1989). *Bayesian Statistics: Principles, Models, and Applications*. New York, NY, USA: John Wiley & Sons.
- Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications In Speech Recognition. *Proceedings of the IEEE*, 77(2), p. 257-285.
- Ramsey, F. P. (1931). *The Foundations of Mathematics and other Logical Essays*. London, UK: Kegan Paul, Trench, Trubner & Co.
- Savage, L. J. (1972). *The Foundations of Statistics*. New York, NY, USA: Dover Publications (originally published in 1954).
- Schmidt-Schauß, M., & Smolka, G. (1991). Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48(1), 1-26.
- Schum, D. A. (1994). *Evidential Foundations of Probabilistic Reasoning*. New York, NY, USA: Wiley.
- Sebastiani, F. (2002). Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34(1), 1-47.
- Shachter, R. D., & Peot, M. A. (1990). Simulation Approaches to General Probabilistic Inference on Bayesian Networks. In M. Henrion, R. D. Shachter, L. N. Kanal & J. F. Lemmer (Eds.), *Uncertainty in Artificial Intelligence 5*. New York, NY: Elsevier Science Publishing Company, Inc.
- Shafer, G. (1976). *A Mathematical Theory of Evidence*. Princeton, NJ, USA: Princeton University Press.
- Shafer, G. (1986). The Construction of Probability Arguments. *Boston University Law Review*, 66(3-4), 799-816.

- Shenoy, P. P., & Demirer, R. (2001). Sequential Valuation Networks for Asymmetric Decision Problems. *European Journal of Operations Research: School of Business, University of Kansas*. Lawrence, KS, USA.
- Shortliffe, E. H., Rhame, F. S., Axline, S. G., Cohen, S. N., Buchanan, B. G., Davis, R., et al. (1975, August). MYCIN: A Computer Program Providing Antimicrobial Therapy Recommendations. *Clinical Medicine*, 34.
- Siolas, G., & d'Alché-Buc, F. (2000, July 24-27). Support Vector Machines Based on a Semantic Kernel for Text Categorization. Paper presented at the International Joint Conference on Neural Networks. Como, Italy.
- Smith, B. (2004). Ontology: Philosophical and Computational. Retrieved January 15, 2005, from <http://ontology.buffalo.edu/smith//articles/ontologies.htm>
- Sowa, J. F. (2000). *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Pacific Grove, CA, USA: Brooks/Cole.
- Spiegelhalter, D. J., Franklin, R., & Bull, K. (1989). Assessment, Criticism, and Improvement of Imprecise Probabilities for a Medical Expert System. In *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence*, pages 285-294. Mountain View, CA.
- Spiegelhalter, D. J., Thomas, A., & Best, N. (1996). Computation on Graphical Models. *Bayesian Statistics*, 5, 407-425.
- Spyns, P., Meersman, R., & Jarrar, M. (2002). Data Modeling versus Ontology Engineering. *SIGMOD Record*, 31(4), 12-17.
- Stone, L. D., Barlow, C. A., & Corwin, T. L. (1999). *Bayesian Multiple Target Tracking*. Boston, MA, USA: Artech House.
- Takikawa, M., d'Ambrosio, B., & Wright, E. (2002). Real-Time Inference with Large-Scale Temporal Bayes Nets. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI-2002)*, pages 477-484, August 1-4. University of Alberta. Edmonton, Alberta, Canada.

- Toffler, A. (1980). *The Third Wave*, 1<sup>st</sup> edition. New York, NY, USA: Morrow.
- Uschold, M., & Gruninger, M. (1996). Ontologies: Principles, Methods, and Applications. *Knowledge Engineering Review*, 11(2), 93-155.
- von Mises, R. (1981). *Probability, Statistics and Truth*. 2<sup>nd</sup> edition. New York, NY, USA: Dover Publications. Originally published in 1928.
- Watson, S. R., & Buede, D. M. (1987). *Decision Synthesis: The Principles and Practice of Decision Analysis*. Cambridge, UK: Cambridge University Press.
- Weatherford, R. (1982). *Philosophical Foundations of Probability Theory*. London, UK: Routledge & K. Paul.

## Appendix A Source Code for The Starship Model

The source code listed below refers to the probabilistic model that was developed for this research effort. It corresponds to four separate files, namely:

- ✓ Starship\_main.spi – It's the execution manager for the Starship model
- ✓ Starship\_framedefs.spi – Defines the model's frame structure
- ✓ Starship\_functions.spi – Defines the functions used in the model
- ✓ Starship\_exec.spi – Create instances and built an SSBN

The output of the models is a Netica file that will be saved in each model's respective folder and will be named as Starship\_v00\_SSN\_00t\_00f\_00e\_00c.dne, where:

- v00 - the model version
- 00t - number of time steps
- 00f - number of friend starships
- 00e - number of enemy starships
- 00c - number of enemy starships with cloak mode

This source code was generated using the following configuration:

### Hardware:

- Apple PowerMac Dual G5 – 2.0 GHz – 1.5 GB RAM

### Software:

- Apple Mac OS X Panther (version 10.3.9)
- Java virtual machine (version 1.4.2\_05)
- IET Quiddity\*Suite (version 4.1.5– build Unix-041217T1653)
- JEdit (version 4.2)
- Norsys Netica (version 2.17) running on top of MS Virtual PC (version 7.0)

### **Starship\_main.spi**

```
# STARSHIP MODEL
```

```

#
# This file is part of the MEBN model inspired in the
# television series Star Trek. The model was used in
# the PhD research of Paulo Costa and in the paper
# "MEBN without Multi-tears"
#
# Authors:
#     Paulo Cesar G da Costa
#     Kathryn B Laskey
#
# The model is composed of the following parts:
#   Starship_main.spi - It's the execution manager for the Starship model
#   Starship_framedefs.spi - Defines the model's frame structure
#   Starship_functions.spi - Defines the functions used in the model
#   Starship_ssbns.spi - Create instances and built an SSBN
#
# <<<<<----- STARSHIP_MAIN.SPI ----->>>>>
#
# version_main=v02; #defines which version of the model this file belongs to.
# The actual definition is below.
#
# This file controls the overall execution of the model
#
# The following lines set the path for the model extensions and the
# file Starship_Main.spi
#

frameSystem();

#
# In order to set the path, copy these two lines in Quiddity prompt and press enter:
#
version_main=v02;
StarshipPath="/Users/pc/PC_PhD_Stuff/MEBNwoTears/Starship_"+version_main+"/";

#
# To run this file you have to type the following at Quiddity's prompt:
# load(StarshipPath+"Starship_main.spi");
#
# Alternatively, you can execute this file inside
# Quiddity*Debugger). To do so, delete the comment
# tag on the path definition line above
#
# <<<<<----- INITIAL SETUP ----->>>>>
#

$qv = javaClass("com.quiddity.visualizer.QuiddityVisualizer")(currentFrameSystem(),false);
$qv->setGoHomeAfterChange(true);
$qv->show();

load(StarshipPath+"Starship_frameDefs.spi");
load(StarshipPath+"Starship_functions.spi");
#
# verify version consistency between the files
#

```

```

if version_main==version_frames then puts("\nStarship_frameDefs.spi file is version "+version_frames+",
consistency check passed!\n");
else puts("\n***** ATTENTION ***** ----->>> CONSISTENCY CHECK FAILED FOR FILE
Starship_frameDefs.spi!!!! \n\n\n\n");
end;
if version_main==version_functions then puts("Starship_functions.spi file is version "+version_functions+",
consistency check passed!\n\n");
else puts("\n***** ATTENTION ***** ----->>> CONSISTENCY CHECK FAILED FOR FILE
Starship_functions.spi!!!! \n\n\n\n");
end;
computesInLog=true; # Computes with logs to avoid underflow error

puts("Support files loaded successfully.\n\n");
puts("Establishing the model parameters...\n");

# <<<<----- /INITIAL SETUP ----->>>>
#
# <<<<----- VARIABLE SETUP ----->>>>
#
# These are the variable that will define the main parameters
# for executing the model. All variables defined in this file
# begin with the preamble "main"

mainTimeSteps = 1; # number of time steps, which
# is also the number of Magnetic Disturbance reports
mainZone = zN_DeepSpace; # define the nature of the zone
mainFShips=1; #number of friendly Starships
mainEShips=3; #number of enemy Starships
mainCloakMode=1; #number of starships in cloak mode.
# In this model, starships in cloak mode are assumed to be
# enemy starships, which includes starships operated by
# neutral/friendly species with intention to harm OwnShip.
# Therefore, the number of starships in cloak mode must
# be smaller than mainEShips

# the procedure below is just a check that we made the
# correct definitions
if (mainEShips < mainCloakMode) then
puts("Error in the variable set: the number of\n");
puts("enemies in cloak mode was set to be bigger\n");
puts("than the number of enemies itself.\n");
puts("\nEdit the file Starship_main.spi and\n");
puts("\nchange either of the variables.\n");
exit();
else puts("variables successfully set.\nWe have defined ");
puts(mainTimeSteps, " time steps in a ", mainZone," area.\n");
puts("With ", mainEShips," enemies and ", mainFShips, " friend or neutral starships nearby.\n");
puts("From the enemy starships, ", mainCloakMode, " (is/are) in cloak mode.\n");
puts(".....\n\n");
end;
#
# <<<<----- /VARIABLE SETUP ----->>>>
#
# <<<<----- MODEL EXECUTION ----->>>>

load(StarshipPath+"Starship_exec.spi");

```

```

if version_main==version_exec then puts("\nStarship_exec.spi file is version "+version_exec+", consistency
check passed!\n");
else puts("\n***** ATENTION ***** ----->>> CONSISTENCY CHECK FAILED FOR FILE
Starship_exec.spi!!!! \n\n\n");
end;

```

```

$qv->updateDisplay();
# Save as Netica model

```

```

saveNetica(StarshipPath+"Starship_"+version_main+"_SSBN_"+ mainTimeSteps+ "t_" +mainFShips + "f_" +
mainEShips + "e_" + mainCloakMode+ "c.dne");

```

```

#<<<<----- /MODEL EXECUTION ----->>>>
#
# <<<<----- /STARSHIP_MAIN.SPI ----->>>>

```

### **Starship framedefs.spi**

```

#      STARSHIP MODEL
#
# This file is part of the MEBN model inspired in the
# Paramount series Star Trek. The model was used in
# the PhD research of Paulo Costa and in the paper
# "MEBN without Multi-tears"
#
# Authors:
#      Paulo Cesar G da Costa
#      Kathryn B Laskey
#
# The model is composed of the following parts:
# Starship_main.spi - It's the execution manager for the Starship model
# Starship_framedefs.spi - Defines the model's frame structure
# Starship_functions.spi - Defines the functions used in the model
# Starship_ssbns.spi - Create instances and built an SSBN
#
# <<<<<<----- STARSHIP_FRAMEDEFS.SPI ----->>>>>
#
version_frames = v02; #defines which version of the model this file belongs to.
#
# This file defines the frame structure that is used to build the SSBN
#
# <<<<----- FRAME DEFINITIONS ----->>>>
#
puts("\nBuilding the Frame System.....\n");

# <<<<----- Frame Zone ----->>>>
#
# <MEBNUID> </MEBNUID>
# <NodeType> TITLE </NodeType>
# <NodeMFrags> Zone </NodeMFrags>
# <NodeHomeMFrags> Zone </NodeHomeMFrags>
# <NodeDistType> </NodeDistType>
# <NodeDescription>
# The instances of the Zone MFrags (i.e. the copies of it that
# were made during the SSBN construction) are the possible

```



```

# space zones in which Enterprise (i.e. OwnStarship) can be
# navigating at a given time.
# </NodeDescription>
# <NodeDist> </NodeDist>
# <QuiddityName> Zone </QuiddityName>
# <QuiddityObj>frame</QuiddityObj>
#
frame Zone isa Frame

# <----- Slot starship ----->
#
# This is a version 2 Tweak to allow multiple cloakMode
# parents for zoneMD.

slot starship
    facet domain = Starship
    facet distribution = UniformDiscreteDistribution

slot anyStInCloakMode
    facet domain = [false, true]
    facet parents = [starship.cloakMode]
    facet distribution = MaxDistribution

# <----- /Slot starship ----->
#
# <----- Slot zoneNature ----->
#
# <MEBNUID> !ZN </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFrag> Zone </NodeMFrag>
# <NodeHomeMFrag> Zone </NodeHomeMFrag>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# A zone can be either a deep space, a planetary
# system, or the boundary of a Black Hole.
# We assumed that a OwnStarship, when in operation (i.e.
# using its decision system), has 80% chance of being
# traveling in a Deep Space Zone, 15% in a Planetary
# System and 5% in the Boundaries of a Black Hole.
# In our model, Black Hole Boundaries are preferred places
# for ambushes from attacking starships with cloaking
# devices, since the high magnetic turbulence generated
# in those zones makes it very hard to even the most
# advanced sensors to distinguish it from the magnetic
# disturbance created by a cloaking device.
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComments> </NodeDistComments>
# <QuiddityName>zoneNature </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>
#
slot zoneNature
    facet domain = [zN_BlackHoleBoundary, zN_DeepSpace, zN_PlanetarySystem]
    facet distribution = [.05, .80, .15]

# <----- /Slot zoneNature ----->

```

```

#
# <----- Slot zoneEShips ----->
#
# <MEBNUID> IZES </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFragment> Zone </NodeMFragment>
# <NodeHomeMFragment> Zone </NodeHomeMFragment>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# This RV establishes the relationship between a given
# zone and the likelihood of having enemy starships
# within OwnStarship's sensor range.
# In other words, it is the probable number of enemy
# ships into sensor range we assume to find in a given
# zone. This means we consider that exists a prior
# probability of finding an enemy starship given
# the nature of the zone in which OwnStarship is
# Navigating through.
# In this model, we restrained the infinitely possible
# number of starships to only five states. That is, we
# assume that it is unlikely to find four or more
# hostile ships in that area, so most of the probability
# distribution mass for this RV will be restricted to the
# states None, One, Two, and Three, while the
# remaining probability will be restricted to the
# aggregating state MoreThan3.
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComments>
# We assume that enemies are likely to be found in
# places that would facilitate an ambush. As an
# example, in a Black Hole Boundary, in which the
# Magnetic Sensor becomes less than helpfull to detect
# Cloak Mode related magnetic disturbances, the
# chances of finding at least one enemy will be 10%,
# which is five times the chance of finding friendly
# starships in the same area.
# </NodeDistComments>
# <QuiddityName>zoneEShips </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>
#
slot zoneEShips
    facet domain = [zES_0, zES_1, zES_2, zES_3, zES_MoreThan3]
    facet parents = [zoneNature]
    facet distribution = function zn {
        zN_BlackHoleBoundary: [.95, .03, .01, .007, .003];
        zN_DeepSpace: [.98, .01, .007, .002, .001];
        zN_PlanetarySystem: [.60, .15, .12, .08, .05];
    }

# <----- /Slot zoneEShips ----->
#
# <----- Slot zoneFShips ----->
#
# <MEBNUID> IZFS </MEBNUID>
# <NodeType> Resident </NodeType>

```

```

# <NodeMFRag> Zone </NodeMFRag>
# <NodeHomeMFRag> Zone </NodeHomeMFRag>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# This RV establishes the relationship between a given
# zone and the likelihood of having friendly starships within
# OwnStarship's sensor range.
# Following the very same rationale of slot zoneEShips
# (node ZoneEShips), we assume that there is a prior
# probability in the number of friendly or neutral
# starships to appear into OwnStarship's sensor range
# given the nature of the zone it is navigating into.
# </NodeDescription>
# <NodeDist> see table </NodeDis>
# <NodeDistComments>
# We assume that unlike enemies, friendly starships
# do not care about being in places suitable for an
# ambush. Therefore, its probability distribution will
# reflect this fact. As an example, in a Black Hole
# Boundary, the chances of finding at least one
# friendly starship are five times smaller than to find
# an ambushing enemy vessel.
# </NodeDistComments>
# <QuiddityName>zoneFShips </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>
#

slot zoneFShips
    facet domain = [zFS_0, zFS_1, zFS_2, zFS_3, zFS_MoreThan3]
    facet parents = [zoneNature]
    facet distribution = function zn {
        zN_BlackHoleBoundary:  [.99, .005, .0035, .001, .0005];
        zN_DeepSpace:  [.98, .01, .007, .002, .001];
        zN_PlanetarySystem:  [.50, .20, .15, .10, .05];
    }

# <----- /Slot zoneFShips ----->
#
# <----- Slot zoneMD ----->
#
# <MEBNUID> !ZMD </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFRag> Zone </NodeMFRag>
# <NodeHomeMFRag> Zone </NodeHomeMFRag>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# ZoneMD(z, t) assesses the value of the magnetic
# disturbance in Zone "z" at the current TimeStep "t".
# This value is influenced by the MD in the previous
# TimeStep (tprev), the fact of whether there is or
# there is not a starship in cloak mode nearby, and the
# nature of the space zone in which the starship is
# located.
# The input node t=!T0 is used to "anchor" the time
# recursion.
# </NodeDescription>

```

```

# <NodeDist> see table </NodeDist>
# <NodeDistComment>
# In a recursive MFrags it is necessary to have a way to
# ground out the recursion. This is similar to a recursive
# algorithm such as computing x!. The definition of x! is:
# 0! = 1
# x! = x(x-1)! for x > 0
# Similarly, in a recursive node it is necessary to define
# the initial distribution for the dynamic BN node and then
# define the next distribution as a function of the previous
# distribution. Here is one way to do it:
# 1. Prev(0) = 0 and Prev(t) = t-1 for t>0
# 2. This creates a problem in this MFrags because now
# DistFromOwn(st,0) has itself as a parent! But really
# this isn't a problem because t=0 is also a parent
# and we define the distribution so that it depends
# on the previous value only when t>0.
# In this case, we assigned to T0 the very same distribution
# we used in the other TimeSteps.
# Regarding the distribution itself, our intention was to make
# background disturbance a somewhat steady phenomena,
# thus the probabilities do not change with time. However,
# if cloack mode is true for any starship nearby Enterprise
# then the next step will be more likely to change the
# intensity of the disturbance, mimicking an unstable
# phenomena.
# </NodeDistComment>
# <QuiddityName>zoneMD </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>

slot zoneMD
    facet domain = [zMD_Low, zMD_Medium, zMD_High]
    facet parents = [zoneNature, anyStInCloakMode, zoneMD.PREV]
    facet initialState = [.70, .20, .10]
    facet distribution = function zn, ascm, znp {
        switch zn {
            zN_BlackHoleBoundary:
                switch ascm{
                    true:
                        switch znp{
                            zMD_Low: [.02, .05, .93];
                            zMD_Medium: [.08, .04, .88];
                            zMD_High: [.12, .18, .70];
                        };
                    false: [.07, .11, .82];
                };
            zN_DeepSpace:
                switch ascm{
                    true:
                        switch znp{
                            zMD_Low: [.70, .18, .12];
                            zMD_Medium: [.80, .05, .15];
                            zMD_High: [.83, .15, .02];
                        };
                    false: [.85, .10, .05];
                };
        }
    }

```

```

        zN_PlanetarySystem:
            switch ascm{
                true:
                    switch znp{
                        zMD_Low:          [.15, .37, .48];
                        zMD_Medium:      [.30, .20, .50];
                        zMD_High:        [.26, .39, .35];
                    };
                false:  [.25, .30, .45];
            };
    }

#
# <----- /Slot zoneMD ----->
#
end;
puts("Frame Zone defined...\n");

# <<<----- /Frame Zone ----->>>
#
# <<<----- Frame Starship ----->>>

# frame Starship: according to the Treknology Encyclopedia L-Z
# (http://www.ex-astris-scientia.org/treknology2.htm#s)
# Starship is the designation for a large type of space vessel
# with warp drive. A starship typically consists of more than one
# deck and has separate departments such as the bridge, engineering
# or sickbay.
# In our model, we use this word to designate any space vessel
#
frame Starship isa Frame
#
# <----- Slot starshipZone ----->

# <MEBNUID> !Scontext_IsAZone_st </MEBNUID>
# <NodeType> Context </NodeType>
# <NodeMFRag> Starship </NodeMFRag>
# <NodeHomeMFRag> IsA </NodeHomeMFRag>
# <NodeDistType> NIL </NodeDistType>
# <NodeDescription>
# In MEBN models, this context node is satisfied when
# the variable "z" is replaced with a unique identifier
# of an entity that has Type equal to "Zone".
# In a Quiddity model, this node is translated to a
# slot named "starshipZone" which has the Frame "Zone"
# as its domain and works as a "pointer" to slots from
# Frame "Zone" that are parents from slots in the
# current frame ("Starship").
# </NodeDescription>
# <NodeDist> NIL </NodeDist>
# <QuiddityName> starshipZone </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>

slot starshipZone

```

```

facet domain = Zone

# <----- /Slot zone ----->
#

# <----- Slot exists ----->
#
# <MEBNUID> IE </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMfrag> Starship </NodeMfrag>
# <NodeHomeMfrag> Starship Existence </NodeHomeMfrag>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# It is the probatility of existence for Starships
# it is a useful way of conveying hypothetical instances
# of a Starship
# Since there is a prior probability of finding enemy or
# friendly starships depending on where OwnShip is
# navigating, these parameters will also influence
# the prior probability of existence. Thus zone.eShips
# and zone.fShips are parents to slot exists
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComment>
# </NodeDistComment>
# <QuiddityName> exists </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>
#

    slot exists
        facet domain = ExistsDomain
        facet parents = [starshipZone.zoneFShips, starshipZone.zoneEShips]
        facet distribution = function fs, es {
            switch fs {
                zFS_0: switch es{
                    zES_0: [1, 0];
                    zES_1: [.5, .5];
                    zES_2: [.33, .67];
                    zES_3: [.25, .75];
                    zES_MoreThan3: [.20, .80];
                };
                zFS_1: switch es{
                    zES_0: [.5, .5];
                    zES_1: [.33, .67];
                    zES_2: [.25, .75];
                    zES_3: [.20, .80];
                    zES_MoreThan3: [.17, .83];
                };
                zFS_2: switch es{
                    zES_0: [.33, .67];
                    zES_1: [.25, .75];
                    zES_2: [.20, .80];
                    zES_3: [.17, .83];
                    zES_MoreThan3: [.20, .80];
                };
                zFS_3: switch es{
                    zES_0: [.25, .75];

```

```

zES_1: [.20, .80];
zES_2: [.17, .83];
zES_3: [.14, .86];
zES_MoreThan3: [.13, .87];
};
zFS_MoreThan3: switch es{
zES_0: [.20, .80];
zES_1: [.17, .83];
zES_2: [.14, .86];
zES_3: [.13, .87];
zES_MoreThan3: [.11, .89];
};
}
}

#
# <----- Slot exists ----->
#
# <----- Slot OpSpec ----->
#
# <MEBNUID> !E </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFRag> Starship </NodeMFRag>
# <NodeHomeMFRag> Starship </NodeHomeMFRag>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# This node conveys the information on what species is
# operating a given starship. Its distribution is derived
# from the number of Friendly and Enemy Starships
# in the vicinity.
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComment>

# </NodeDistComment>
# <QuiddityName> exists </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>
## slot shipType: conveys the nature of a starship. In other
# words, whether we are dealing with an enemy or a friendly
# starship (which also includes neutral starships)
#
slot opSpec
  facet domain = [oS_Cardassian, oS_Friend, oS_Klingon, oS_Romulan, oS_Unknown, absurd]
  facet parents = [exists, starshipZone.zoneFShips, starshipZone.zoneEShips]
  facet distribution = function ex, fs, es {
    if (ex==Context.OUT) then [0, 0, 0, 0, 0, 1]
    else switch fs {
      zFS_0: switch es{
        zES_0: [0, 0, 0, 0, 0, 1];
        zES_1: [.50, 0, .15, .30, .05, 0];
        zES_2: [.45, 0, .15, .30, .10, 0];
        zES_3: [.40, 0, .20, .28, .12, 0];
        zES_MoreThan3: [.35, 0, .23, .27, .15, 0];
      };
      zFS_1: switch es{
        zES_0: [0, .50, .30, .15, .05, 0];
        zES_1: [.30, .30, .20, .10, .10, 0];

```

```

                                zES_2:          [.25, .20, .28, .15, .12, 0];
                                zES_3:          [.25, .18, .16, .26, .15, 0];
                                zES_MoreThan3:  [.23, .18, .18, .26, .15, 0];
                                };
zFS_2: switch es{
                                zES_0:          [0, .50, .30, .10, .10, 0];
                                zES_1:          [.20, .25, .15, .28, .12, 0];
                                zES_2:          [.27, .27, .18, .13, .15, 0];
                                zES_3:          [.27, .25, .19, .14, .15, 0];
                                zES_MoreThan3:  [.26, .24, .20, .15, .15, 0];
                                };
zFS_3: switch es{
                                zES_0:          [0, .55, .20, .10, .15, 0];
                                zES_1:          [.18, .25, .16, .26, .15, 0];
                                zES_2:          [.25, .27, .14, .19, .15, 0];
                                zES_3:          [.25, .25, .20, .15, .15, 0];
                                zES_MoreThan3:  [.25, .23, .20, .17, .15, 0];
                                };
zFS_MoreThan3: switch es{
                                zES_0:          [0, .55, .20, .10, .15, 0];
                                zES_1:          [.18, .23, .26, .18, .15, 0];
                                zES_2:          [.24, .26, .15, .20, .15, 0];
                                zES_3:          [.23, .25, .17, .20, .15, 0];
                                zES_MoreThan3:  [.23, .25, .22, .15, .15, 0];
                                };
                                }
                                }
                                end;
                                }

#
# <----- Slot /OpSpec ----->
#
# <----- Slot starshipClass ----->
#
## <MEBNUID> ISC </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFrags> Starship </NodeMFrags>
# <NodeHomeMFrags> Starship </NodeHomeMFrags>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# This RV assesses what is the class of the
# starship represented by "st". It is influenced by the
# kind of species that is operating the starship and the
# very own existence of the starship itself (as defined
# in the context node "exists").
# There is a vast literature of classes and subclasses of
# starships for each species (e.g. see
# http://techspecs.acalltoduty.com).
# However, for this simple model we used a general
# taxonomy that aggregates the starships in five different
# classes (WarBird, Cruiser, Explorer, Frigate and
# Freighter).
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComment>
# </NodeDistComment>
# <QuiddityName> starshipClass </QuiddityName>

```



```

# <QuiddityObj>slot</QuiddityObj>

slot starshipClass
  facet domain = [sC_WarBird, sC_Cruiser, sC_Explorer, sC_Frigate, sC_Freighter, absurd]
  facet parents = [exists, opSpec]
  facet distribution = function e, os {
    if (e==Context.OUT) then [0, 0, 0, 0, 0, 1]
    else switch os {
      oS_Cardassian: [.40, .10, 0, .40, .10, 0];
      oS_Friend:      [.10, .30, .25, .20, .15, 0];
      oS_Klingon:     [.25, .50, .15, 0, .10, 0];
      oS_Romulan:     [.60, 0, .30, 0, .10, 0];
      oS_Unknown:     [.10, .10, .35, .10, .35, 0];
      absurd:         [0, 0, 0, 0, 0, 1];
    }
  }
end;
}

# <----- Slot /starshipClass ----->
#
# <----- Slot cloakMode ----->
#
# <MEBNUID> ICM </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFRag> Starship </NodeMFRag>
# <NodeHomeMFRag> Starship </NodeHomeMFRag>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# This is a boolean variable that defines whether
# the starship in question is in cloak mode.
# In our model, we assume that only Romulan and
# Klingon starships can be in cloak mode, since the
# Federation still does not have such technology
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComment>
# Only Romulans and Klingons have the technology.
# However, Klingons would use it only when having
# aggressive intentions against OwnShip (i.e. breaking
# the peace treaty), so we expect them to use it less
# than the Romulans.
# Unknown species might have acquired the technology,
# but that is not very likely.
# </NodeDistComment>
# <QuiddityName> cloakMode </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>#
#
slot cloakMode
  facet domain = [false, true]
  facet parents = [opSpec, starshipClass]
  facet distribution = function os, sc {
    if os==absurd then [1, 0]
    else {
      switch os {
        oS_Cardassian: switch sc {
          sC_WarBird: [1, 0];

```

```

        sC_Cruiser:      [1, 0];
        sC_Explorer:    [1, 0];
        sC_Frigate:     [1, 0];
        sC_Freighter:[1, 0];
        absurd:         [1, 0];
    };
oS_Friend: switch sc {
    sC_WarBird:      [1, 0];
    sC_Cruiser:      [1, 0];
    sC_Explorer:    [1, 0];
    sC_Frigate:     [1, 0];
    sC_Freighter:[1, 0];
    absurd:         [1, 0];
};
oS_Klingon: switch sc {
    sC_WarBird:      [.65, .35];
    sC_Cruiser:      [.65, .35];
    sC_Explorer:    [.95, .05];
    sC_Frigate:      [.85, .15];
    sC_Freighter: [.99, .01];
    absurd:         [1, 0];
};
oS_Romulan: switch sc {
    sC_WarBird: [.10, .90];
    sC_Cruiser:  [.20, .80];
    sC_Explorer: [.40, .60];
    sC_Frigate:  [.30, .70];
    sC_Freighter:[.80, .20];
    absurd: [1, 0];
};
oS_Unknown: switch sc {
    sC_WarBird: [.99, .01];
    sC_Cruiser:  [.99, .01];
    sC_Explorer: [.995, .005];
    sC_Frigate:  [.995, .005];
    sC_Freighter: [.9995, .0005];
    absurd: [1, 0];
};
};
    }
end;
}

# <----- /Slot cloakMode ----->
#
# <----- Slot distFromOwn ----->
#
# <MEBNUID> !DFO </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFrags> Starship </NodeMFrags>
# <NodeHomeMFrags> Starship </NodeHomeMFrags>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# This RV assesses the distance from a starship "st" to
# OwnStarship at TimeStep "t". This distance is measured
# according to weapon's ranges, since its main purpose is

```

```

# to assess the ability to any given starship to harm OwnShip.
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComment>
# </NodeDistComment>
# <QuiddityName> distFromOwn </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>#
#

    slot distFromOwn
        facet domain = [dFO_OutOfRange, dFO_TorpedoRange, dFO_Phaser2Range,
dFO_Phaser1Range, dFO_PulseCanonRange, absurd]
        facet parents = [exists, distFromOwn.PREV]
        facet distribution = function e, dfo {
            if e==Context.OUT then [0, 0, 0, 0, 0, 1]
            else switch dfo{
                dFO_OutOfRange:          [.60, .30, .05, .04, .01, 0];
                dFO_TorpedoRange:        [.25, .40, .25, .07, .03, 0];
                dFO_Phaser2Range:        [.06, .25, .40, .25, .04, 0];
                dFO_Phaser1Range:        [.03, .07, .25, .40, .25, 0];
                dFO_PulseCanonRange:     [.01, .04, .10, .35, .50, 0];
                absurd:                  [0, 0, 0, 0, 0, 1];
            }
        }
        end;
    }
    facet initialState = [.1936, .2245, .2233, .2156, .1430, 0]

# <----- /Slot distFromOwn ----->
#
#
# <----- Slot harmPotential ----->
#
# <MEBNUID> !HP </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFrags> Starship </NodeMFrags>
# <NodeHomeMFrags> Starship </NodeHomeMFrags>
# <NodeDistType> Netica Table </NodeDistType>
# <NodeDescription>
# This RV assesses the potential of starship "st" to harm
# OwnShip at current TimeStep "t". It is based on the
# starship weapons' range (based on its class) and its
# distance from OwnShip.
# It is important to note that here we are not assessing
# *intention* to harm, but only *ability* to do so. Therefore,
# even friendly starships can have HarmPotential with
# value true (e.g. provide that they are within their
# respective weapons range).
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComment>
# </NodeDistComment>
# <QuiddityName> harmPotential </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>#
#

    slot harmPotential
        facet domain = [false, true]

```

```

facet parents = [distFromOwn, starshipClass]
facet distribution = function dfo, sc {
  switch dfo{
    dFO_OutOfRange:      switch sc {
      sC_WarBird:        [1, 0];
      sC_Cruiser:         [1, 0];
      sC_Explorer:        [1, 0];
      sC_Frigate:          [1, 0];
      sC_Freighter:       [1, 0];
      absurd:             [1, 0];
    };
    dFO_TorpedoRange:     switch sc {
      sC_WarBird:         [0, 1];
      sC_Cruiser:          [.20, .80];
      sC_Explorer:         [.70, .30];
      sC_Frigate:          [.90, .10];
      sC_Freighter:       [.99, .01];
      absurd:             [1, 0];
    };
    dFO_Phaser2Range:     switch sc {
      sC_WarBird:         [0, 1];
      sC_Cruiser:          [0, 1];
      sC_Explorer:         [.40, .60];
      sC_Frigate:          [.60, .40];
      sC_Freighter:       [.95, .05];
      absurd:             [1, 0];
    };
    dFO_Phaser1Range:     switch sc {
      sC_WarBird:         [0, 1];
      sC_Cruiser:          [0, 1];
      sC_Explorer:         [.05, .95];
      sC_Frigate:          [0, 1];
      sC_Freighter:       [.90, .10];
      absurd:             [1, 0];
    };
    dFO_PulseCanonRange:  switch sc {
      sC_WarBird:         [0, 1];
      sC_Cruiser:          [0, 1];
      sC_Explorer:         [0, 1];
      sC_Frigate:          [0, 1];
      sC_Freighter:       [.80, .20];
      absurd:             [1, 0];
    };
    absurd:               [1, 0];
  };
}

#
# <----- /Slot harmPotential ----->
#
#
# <----- Slot dangerToOwnStarship ----->
#
# Paliative solution to handle the problem of many instances of one parent
# to a unique slot.
# In this case, each instance of slots harmPotential and opSpec

```

# are parents of dangerToSelf. Thus, we created a unique slot for  
 # each starship calles dangerToOwnShip, which assesses the potential  
 # danger brought by that specific starship. Then, all the dangerToOwnStarship  
 # slots are aggregated to the dangerToSelf slot in the OwnStarship Frame  
 # using Quiddity's MaxDistribution.

```

    slot dangerToOwnStarship
      facet domain= [ dTS_Low , dTS_Medium, dTS_High, dTS_Unacceptable]
      facet parents= [harmPotential, opSpec]
      facet distribution = function hp, os {
        if hp==false then [1, 0, 0, 0]
        else switch os {
          oS_Cardassian: [0, .02, .08, .90];
          oS_Friend:      [.99, .01, 0, 0];
          oS_Klingon:     [ .65, .20, .10, .05];
          oS_Romulan:     [.03, .09, .18, .70];
          oS_Unknown:     [.20, .30, .30, .20];
          absurd:         [1, 0, 0, 0];
        }
      }
    end;
  }

# <----- /Slot dangerToOwnStarship ----->

end;

puts("Frame Starship defined...\n");

# <<<----- /Frame Starship ----->>>
#
# <<<----- Frame OwnStarship ----->>>
#
# frame OwnStarship:
# This Frame has the sole objective to model the features
# we just one to have in the Enterprise (OwnShip)
# In MEBN, such modeling would be accomplished by the
# use of a node OwnStarship, which could be employed to
# define context of MFrag in which we want to explicitly
# constraint OwnStarship, but Quiddity doesn't have a
# flexible way of imposing these constraints so the best
# approach is to model OwnStarship as a separate frame.
#
frame OwnStarship isa Frame
#
# <----- Slot starship ----->
#
# <MEBNUID> !OSTcontext_IsAStarship_st </MEBNUID>
# <NodeType> Context </NodeType>
# <NodeMFrag> Zone </NodeMFrag>
# <NodeHomeMFrag> IsA </NodeHomeMFrag>
# <NodeDistType> NIL </NodeDistType>
# <NodeDescription>
# In MEBN models, this context node its satisfied when
# the variable "st" is replaced with a unique identifier
# of an entity that has Type equal to "Starship".
# In a Quiddity model, this node is translated to a

```

```

# slot named "starship" which has the Frame "Starship"
# as its domain and works as a "pointer" to slots from
# Frame "Starship" which are parents from slots in the
# current frame ("OwnStarship").
# </NodeDescription>
# <NodeDist> NIL </NodeDist>
# <QuiddityName>starship </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>

slot starship
    facet domain = Starship
    facet distribution = UniformDiscreteDistribution

# <----- /Slot starship ----->

# <----- Slot zone ----->

# <MEBNUID> !Scontext_IsAZone_st </MEBNUID>
# <NodeType> Context </NodeType>
# <NodeMFragment> Starship </NodeMFragment>
# <NodeHomeMFragment> IsA </NodeHomeMFragment>
# <NodeDistType> NIL </NodeDistType>
# <NodeDescription>
# In MEBN models, this context node is satisfied when
# the variable "z" is replaced with a unique identifier
# of an entity that has Type equal to "Zone".
# In a Quiddity model, this node is translated to a
# slot named "ownStarshipZone" which has the Frame "Zone"
# as its domain and works as a "pointer" to slots from
# Frame "Zone" that are parents from slots in the
# current frame ("OwnStarship").
# </NodeDescription>
# <NodeDist> NIL </NodeDist>
# <QuiddityName> ownStarshipZone </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>

slot ownStarshipZone
    facet domain = Zone

# <----- /Slot zone ----->
#
# <----- Slot dangerToSelf ----->
#
# <MEBNUID> !DTS </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFragment> DangerToSelf </NodeMFragment>
# <NodeHomeMFragment> DangerToSelf </NodeHomeMFragment>
# <NodeDistType> Pseudo-Code </NodeDistType>
# <NodeDescription>
# This node assesses the level of danger to which
# OwnStarship "s" is exposed at a given time "t".
# Basically, this danger level will be a function of the ability of
# a starship "st" to harm OwnStarship and of the intention
# of whoever is operating starship "st" to harm OwnStarship,
# the latter being implied from the knowledge of what species
# is operating starship "st".

```

```

# </NodeDescription>
# <NodeDist>
# This distribution cannot be represented by a single table,
# so we convey it via this pseudo-code:
# 1.    distribution [Un, Hi, Me, Lo] = function {
# 2.        for all st in parents(DangerToSelf(s, t)) {
# 3.            if any STi have (OpSpec == Cardassian and HarmPot == true) then
# 4.                [Un = .90 + min( .10; .025*number(STi) ), Hi = (1 - Un) * .8,
# 5.                Me = (1 - Un) * .2, Lo = 0];
# 6.            else if any STj have (OpSpec == Romulan and HarmPot == true) then
# 7.                [Un = .70 + min( .30; .03*number(STj) ), Hi = (1 - Un) * .6,
# 8.                Me = (1 - Hi) * .3, Lo = (1 - Hi) * .1];
# 9.            else if any STj have (OpSpec == Unknown and HarmPot == true) then
# 10.                [Un = (1 - Hi), Hi = .50 - min( .20; .02*number(STk) ),
# 11.                Me = .50 - min( .20; .02*number(STk) ), Lo = (1 - Me)];
# 12.            else if any STk have (OpSpec == Klingon and HarmPot == true) then
# 13.                [Un = 0.10, Hi = 0.15, Me = .15, Lo = .65];
# 14.            else if any STl have (OpSpec == Friend and HarmPot == true) then
# 15.                [Un = 0, Hi = 0, Me = .01, Lo = .99];
# 16.            else [Un = 0, Hi = 0, Me = 0, Lo = 1];
# 17.        }
# 18.    }
# </NodeDist>
# <NodeDistComment>
# Note that there can be many starships "st" at a given
# time t. Thus, the probability distribution has to take into
# consideration the hypothesis of having many starships
# with the potential to harm OwnStarship.
# </NodeDistComment>
# <QuiddityName> dangerToSelf </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>#
#
# NOTE: for this specfic version of the MTheory in the MTears paper
# we added an intermediary dangerToOwnStarship in each starship instance
# which we now aggregate to this node by the use of MaxDistribution
# In other words, the distribution stated above is not actually represented
# here.
#
#         slot dangerToSelf
#             facet domain= [ dTS_Low, dTS_Medium, dTS_High, dTS_Unacceptable]
#             facet parents= [starship.dangerToOwnStarship]
#             facet distribution = MaxDistribution

# <----- /Slot dangertoSelf ----->
#
# <----- Slot dangerToOthers ----->
#
# <MEBNUID> !DTO </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFRag> DangerToOthers </NodeMFRag>
# <NodeHomeMFRag> DangerToOthers </NodeHomeMFRag>
# <NodeDistType> Netica table </NodeDistType>
# <NodeDescription>
# This node conveys the ability of OwnShip "s" to inflict
# danger to another starship "st" at TimeStep "t". It is
# based on OwnShip's Weapons (implicitly considered

```

```

# in the probability distribution) and its distance from
# starship "st".
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComment> </NodeDistComment>
# <QuiddityName> dangerToOthers</QuiddityName>
# <QuiddityObj>slot</QuiddityObj>#
#
    slot dangerToOthers
        facet domain = [false, true]
        facet parents = [starship.distFromOwn]
        facet distribution = function dfo {
            dFO_OutOfRange:          [1, 0];
            dFO_TorpedoRange:        [0, 1];
            dFO_Phaser2Range:        [.02, .98];
            dFO_Phaser1Range:        [.10, .90];
            dFO_PulseCanonRange:     [.20, .80];
            absurd:                  [1, 0];
        }
#
# <----- /Slot dangerToOthers ----->
#
end;

puts("Frame OwnStarship defined...\n");
#
# <<<----- /Frame OwnStarship ----->>>
##
# <<<----- Frame SensorReport ----->>>
#
# For this simple model, instead of
# creating a frame report with subtypes sensor and
# magnetic disturbance, we opted for the simplest
# approach of creating two separate classes.
#

frame SensorReport isa Frame

# <----- Slot subject ----->
#
# <MEBNUID> IS </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFRag> SR Data </NodeMFRag>
# <NodeHomeMFRag> SR Data </NodeHomeMFRag>
# <NodeDistType> Netica table </NodeDistType>
# <NodeDescription>
# This RV has as its possible values all the unique
# identifiers of the entities that can be the subject of
# the sensor report being represented by the variable "sr".
# In this model, Sensor reports can refer to Starships
# (real or hypothetical), in which case the RV will assume
# the unique identifier of that starship as its value, or it can
# refer to nothing (i.e. a spurious report), in which case it
# will assume the unique identifier of a spurious report as
# its value (e.g. O_Spurious)
# </NodeDescription>

```



```

# <NodeDist> see table </NodeDist>
# <NodeDistComments>
# We assigned 4% chances of a spurious report, but this is a
# number that would reflect the receiver characteristics.
# We assumed that a report caused by OwnStarship (here
# assumed as !ST0) falls into the spurious category so we
# assigned 0% probability of !ST0 being the subject of a
# report.
# Finally, we assigned the remaining probability (96%) as
# equally distributed among the Starships.
# </NodeDistComments>
# <QuiddityName>subject </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>
#
#      OBS: the Quiddity translation does not include the
# probabilistic assignment to node subject. Instead, it
# is used as a "pointer" to Frame Starship to the slots
# of Frame SensorReport
#
      slot subject
      facet domain = Starship

# <----- /Slot subject ----->
#
# <----- Slot sRClass ----->
#
# <MEBNUID> !SRC </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFrag> SensorReport </NodeMFrag>
# <NodeHomeMFrag> SensorReport </NodeHomeMFrag>
# <NodeDistType> Netica table </NodeDistType>
# <NodeDescription>
# This RV conveys the result of a sensor report "sr"
# regarding to the class of a given starship at current
# TimeStep "t".
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComments>
# Some remarks regarding the input values:
# Exists(st) - If the starship represented by "st" doesn't
# exists, then the report will be spurious and will have
# equally likely chances of returning any information on class.
# StarshipClass(st) - The ability of a sensor report to match
# the actual class of a starship is a direct consequence of
# the sensor accuracy. For this simple model, we are
# assuming a uniformly distributed accuracy of 95%,
# with the probability of error unevenly dispersed according
# to the similarity of the Starship Classes (e.g. it is easier to
# confuse a Warbird with a Cruiser than with a Freighter).
# CloakMode(st) - If a Starship in Cloak mode generates
# a report for some reason, then its result will be a spurious
# report with equally likely results.
# </NodeDistComments>
# <QuiddityName>sRClass </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>
#

```

```

slot sRClass
    facet domain = [sRC_WarBird, sRC_Cruiser, sRC_Explorer, sRC_Frigate, sRC_Freighter,
sRC_Nothing]
    facet parents = [subject.exists, subject.cloakMode, subject.starshipClass]
    facet distribution = function se, scm, ssc {
        if se==Context.OUT then [0, 0, 0, 0, 0, 1]
        elif scm==true then [0.0001, .0001, .0001, .0001, .0001, .9995]
        else switch ssc {
            sC_WarBird:    [.95, .025, .015, .007, .003, 0];
            sC_Cruiser:    [.02, .95, .015, .01, .005, 0];
            sC_Explorer:   [.01, .015, .95, .015, .01, 0];
            sC_Frigate:    [.008, .012, .02, .95, .01, 0];
            sC_Freighter:  [.01, .01, .02, .01, .95, 0];
            absurd:        [0, 0, 0, 0, 0, 1];
        }
    }
end;
}

#
#
# <----- /Slot sRClass ----->
#
# <----- /Slot sRDistance ----->
#
# <MEBNUID> !SRD </MEBNUID>
# <NodeType> Resident </NodeType>
# <NodeMFrags> SensorReport </NodeMFrags>
# <NodeHomeMFrags> SensorReport </NodeHomeMFrags>
# <NodeDistType> Netica table </NodeDistType>
# <NodeDescription>
# This RV conveys the result of a sensor report "sr"
# regarding to the distance of a given starship to
# OwnStarship at current TimeStep "t".
# </NodeDescription>
# <NodeDist> see table </NodeDist>
# <NodeDistComments>
# Some remarks regarding the input values:
# Exists(st) - If the starship represented by "st" doesn't
# exists, then the report will be spurious and will have
# equally likely chances of returning any information on class.
# DistFromOwn(st, t) - The ability of a sensor report to match
# the actual distance of a starship at any timestep "t" is a
# direct consequence of the sensor's accuracy. For this
# simple model, we are assuming an accuracy of 99%, with
# the probability of error distributed proportionally to the
# adjacent values.
# CloakMode(st) - If a Starship in Cloak mode generates
# a report for some reason, then its result will be a spurious
# report with equally likely results.
# </NodeDistComments>
# <QuiddityName>sRDistance </QuiddityName>
# <QuiddityObj>slot</QuiddityObj>
#
slot sRDistance
    facet domain = [sRD_OutOfRange, sRD_TorpedoRange, sRD_Phaser2Range,
sRD_Phaser1Range, sRD_PulseCanonRange, sRD_Nothing]
    facet parents = [subject.exists, subject.cloakMode, subject.distFromOwn]

```

```

facet distribution = function se, scm, sdfo {
  if se==Context.OUT then [0, 0, 0, 0, 0, 1]
  elif scm==true then [0.0001, .0001, .0001, .0001, .0001, .9995];
  else switch sdfo {
    dFO_OutOfRange:      [.99, .004, .003, .002, .001, 0];
    dFO_TorpedoRange:    [.0035, .99, .0035, .002, .001, 0];
    dFO_Phaser2Range:    [.0015, .0035, .99, .0035, .0015, 0];
    dFO_Phaser1Range:    [.001, .002, .0035, .99, .0035, 0];
    dFO_PulseCanonRange: [.001, .002, .003, .004, .99, 0];
    absurd:              [0, 0, 0, 0, 0, 1];
  }
  end;
}

# <----- /Slot sRDistance ----->
#

end;

puts("Frame SensorReport defined...\n");

# <<<----- /Frame SensorReport ----->>>

puts("Frame System for Starship Basic Model is ready!!!\n\n");
#
# <<<<----- /FRAME DEFINITIONS ----->>>>
#
# <<<<<<----- /STARSHIP_FRAMEDEFS.SPI ----->>>>>>

```

### **Starship functions.spi**

```

#                               STARSHIP MODEL
#
# This file is part of the MEBN model inspired in the
# Paramount series Star Trek. The model was used in
# the PhD research of Paulo Costa and in the paper
# "MEBN without Multi-tears"
#
# Authors:
#         Paulo Cesar G da Costa
#         Kathryn B Laskey
#
# The model is composed of the following parts:
#   Starship_main.spi - It's the execution manager for the Starship model
#   Starship_framedefs.spi - Defines the model's frame structure
#   Starship_functions.spi - Defines the functions used in the model
#   Starship_ssbn.spi - Create instances and built an SSBN
#
# <<<<<<----- STARSHIP_FUNCTIONS.SPI ----->>>>>>
#
version_functions=v02; #defines which version of the model this file belongs to.
#
# This file defines the functions that will be used to build the SSBN
#

```

```

puts("\nStarting to define the functions\n");
puts(".....\n");

# The two functions below were developed by Dr. Kathryn
# Laskey for the Plasma project. They enable the model's
# visualization by means of Quiddity Visualizer.
#
def displayStatus(stpnum) {
  enqueueScript("displayStatusSub(" + stpnum + ")");
};

def displayStatusSub(stpnum) {
  for c in NamedEntity->retrieveInstances(false) {
    c->printDetail();    # display all the instances
  };
  $qv->updateDisplay();
  puts("Paused at " + stpnum + ". Hit <Enter> to continue...");
  while stdin->read() != 10 {}
};

# numberToFShips(fships)
# fships -> the number of FShips in a zone
# The function converts the number to the string format
# that is used in slot zoneFShips

def numberToFShips(fships){
  if (fships == 0) then state = zFS_0
  elif (fships == 1) then state = zFS_1
  elif (fships == 2) then state = zFS_2
  elif (fships == 3) then state = zFS_3
  elif (fships > 3) then state = zFS_MoreThan3
  else ("Error in function numberToFShips\n")
  end;
  state;
};

# numberToEShips(eships)
# eships -> the number of EShips in a zone
# The function converts the number to the string format
# that is used in slot zoneEShips

def numberToEShips(eships){
  if (eships == 0) then state = zES_0
  elif (eships == 1) then state = zES_1
  elif (eships == 2) then state = zES_2
  elif (eships == 3) then state = zES_3
  elif (eships > 3) then state = zES_MoreThan3
  else ("Error in function numberToEShips\n")
  end;
  state;
};
#
# randomDistance()
# Simple random generator that returns a distance that
# can be used in the distFromOwn RV

```

```

# Output states:
# dFO_OutOfRange
# dFO_TorpedoRange
# dFO_Phaser2Range
# dFO_Phaser1Range
# dFO_PulseCanonRange
# It does not return absurd as a value.
#      It *does not* gives a equal likelihood for distances,
# but concentrates more probability to the farther
# distances. The idea is that since this would be used
# for an initial setup, most of the starships would be
# relatively distant of OwnShip. As the situation develops
# then we can have starships closing in
#
def randomDistance(){
    rd = random->nextDouble(); # create a random number
    if (0 <= rd && rd < .05) then rdist= dFO_PulseCanonRange
    elif ( 0.05 <= rd && rd < .15) then (rdist= dFO_Phaser1Range)
    elif ( 0.15 <= rd && rd < .25) then (rdist= dFO_Phaser2Range)
    elif ( 0.25 <= rd && rd < .6) then (rdist= dFO_TorpedoRange)
    elif ( 0.6 <= rd && rd < 1) then (rdist= dFO_OutOfRange)
    else (puts("error in the Distance random generator"))
    end;
    rdist;
};

# randomSpecies(specBehavior)
# Simple random generator that returns an Operator
# Species that can be used in the opSpec RV
# Input:
# specBehavior - variable that has states friend or enemy.
# Each corresponding to a given species current behavior
# with respect to OwnShip
# cloak - defines whether or not the species to be created
# has a cloaking device capability.
# Output states:
# oS_Cardassians
# oS_Friend
# oS_Klingon
# oS_Romulan
# oS_Unknown
# It does not return absurd as a value.
#      It *does not* gives a equal likelihood for distances,
# but concentrates more probability to the farther
# distances. The idea is that since this would be used
# for an initial setup, most of the starships would be
# relatively distant of OwnShip. As the situation develops
# then we can have starships closing in
#
def randomSpecies(specBehavior, cloak){
    rd = random->nextDouble(); # create a random number
    if (cloak==true) then {
        if (specBehavior == friend) then {
            puts("Error: Species with friend behavior do not\n");
            puts("activate cloaking devices\n");
            puts("function randomSpecies has improper input\n");
        }
    }
}

```

```

    }
    elif (specBehavior == enemy) then {
        if (0 <= rd && rd < .9) then (rspec= oS_Romulan)
        elif ( 0.9 <= rd && rd < .99) then (rspec= oS_Klingon)
        elif ( 0.99 <= rd && rd < 1) then (rspec = oS_Unknown)
        elif ( 0.9 <= rd && rd < 1) then (rspec = oS_Unknown)
        else (puts("error in the Species random generator"))
        end;
    }
    else (puts("error in the Species random generator"))
    end;
}
elif (specBehavior == enemy) then {
    if (0 <= rd && rd < .5) then (rspec= oS_Cardassian)
    elif ( 0.5 <= rd && rd < .65) then (rspec= oS_Klingon)
    elif ( 0.65 <= rd && rd < .9) then (rspec = oS_Romulan)
    elif ( 0.9 <= rd && rd < 1) then (rspec = oS_Unknown)
    else (puts("error in the Species random generator"))
    end;
}
elif (specBehavior == friend) then {
    if (0 <= rd && rd < .5) then (rspec= oS_Friend)
    elif ( 0.5 <= rd && rd < .75) then (rspec= oS_Klingon)
    elif ( 0.75 <= rd && rd < .85) then (rspec = oS_Romulan)
    elif ( 0.85 <= rd && rd < 1) then (rspec = oS_Unknown)
    else (puts("error in the Species random generator"))
    end;
}
else (puts("Improperly defined specBehavior in randomSpecies"))
end;
rspec;
};
#
# makeEnemies(nk, ncloak)
# nk -> Total number of enemy starships to be created
# ncloak -> Number of enemy starships that are within
# enterprise's sensor range but in cloak mode.
# The function creates the nk instances of enemy starships and
# returns an array with the form array(enemy),
# where each element is an instance of frame Starship
#
# The intended use for makeEnemies() is to create a set
# of Enemy Starships that, among other possibilities, can
# be used in conjunction with makeFriends() to build a
# Ground Truth set for model experiments

def makeEnemies(z, nk, ncloak) {
    charset=array(0);
    duck = nk-ncloak; # These enemies are not in cloak mode
    for i in duck {
        enduck = Starship->makeInstance();
        enduck->opSpec = randomSpecies(enemy, false);
        enduck->cloakMode=false;
        enduck->starshipZone=z;
        enduck->distFromOwn=randomDistance();
        charset->addElement(enduck);
    }
}

```

```

    };
    for j in ncloak { # These enemies are in cloak mode
        encloak = Starship->makeInstance();
        encloak->opSpec = randomSpecies(enemy, true);
        encloak->cloakMode=true;
        encloak->starshipZone=z;
        encloak->distFromOwn=randomDistance();
        charset->addElement(encloak);
    };
    charset;
};

# makeFriends(nf)
# nf -> Total number of friend or neutral starships to be
# created.
# This function creates nf instances of friend/neutral
# starships and returns an array with the form array(friends)
# where each element is an instance of frame Starship

def makeFriends(zf, nf) {
    charset=array(0);
    for i in nf {
        frd = Starship->makeInstance();
        frd->opSpec = randomSpecies(enemy, false);
        frd->cloakMode=false;
        frd->starshipZone=zf;
        frd->distFromOwn=randomDistance();
        charset->addElement(frd);
    };
    charset;
};

# makeGTSet(nen, nkc, nfr)
# nen -> number of enemies
# nec -> number of enemies in cloak mode
# nfr -> number of friends
# uses makeEnemies() and makeFriends() to create
# a set spaceships.

def makeGTSet(zn, nen, nec, nfr) {
    gt=makeEnemies(zn, nen, nec);
    nicepeople=makeFriends(zn, nfr);
    for i in nfr {
        gt->addElement(nicepeople(i));
    };
    gt;
};

# printCharSet(cs)
# cs -> an array with the format of the output of
# function makeGTSet(nen, nec, nfr)
# Prints the set of Starships to be used as Ground Truth
# that should be contained in the argument cs.

def printCharSet(cs){
    puts("Characteristics of the Starships:\n");

```

```

    for chr in cs {
        st=chr->opSpec->getValue();
        cm=chr->cloakMode->getValue();
        zn=chr->starshipZone->zoneNature->getValue();
        qd=chr->distFromOwn->getValue();
        puts(chr,"\tOperator Species -> ",st, "\tCloak Mode-> ",cm);
        puts("\tZone->",zn, "\tRange -> ",qd,"\n");
    };
    puts("\nEnd of the Starship List!!\n");
};

# buildGTKb(ezn, nts, gts)
# ezr -> area in which Enterprise is in
# nts -> number of time steps
# gts -> array with the ground truth
# During each time step, this function creates one instance
# of frame SensorReport for each starship in the Ground
# Truth set (gts) and reads the zoneMD value, which
# is influenced by the nature of the area and the existence of
# starships in cloak mode.
# Then, it samples and set the values for the slots
# SensorReport.sRClass, SensorReport.sRDistance.
# It also sets the values of those slots with the sampled
# results, which is an optional feature that can be canceled.
# The output is an array in the format:
# (list(zoneMD, stresults)),
#
# Recall that starships in cloak mode doesn't generate
# sensor reports. Yet, we have to consider them too (the
# enemies in cloak mode), since the false alarm rate is a
# bit higher when someone is around with an active cloak
# device. Finally, even if nobody is there, we have to
# consider the false alarm rate as well.

def buildGTKb(ezn, nts, gts) {
    listres=array(0); # saves the results of all time steps
    for i in nts { # for each time step
        stresults=array(0); # saves each starship's results of one time step
        for j in gts{ # for each starship
            snres=SensorReport->makeInstance(); # create an instance of sensor report
            snres->subject=j ; # relate that instance to
one starship
            getsnslot=snres->sRClass; # get this instance's sRClass slot
            spsnslot=sampleNode(getsnslot); # and sample it
            snres->sRClass=spsnslot;
# This line is optional. It sets the value of slot sRClass to be the
# one sampled in spsnslot (see above). If setting the results is not what we
# want to do (i.e. if we prefer to leave it empty) then just ignore the line
# by adding a comment mark in front of it.
            getdtslot=snres->sRDistance; # get this instance's sRDistance slot
            spdtslot=sampleNode(getdtslot); # and sample it
            snres->sRDistance=spdtslot;
# This line is optional. It sets the value of slot sRDistance
# to be the one sampled in spdtslot (see above). If setting the results is not
# what we want to do (i.e. if we prefer to leave it empty) then just ignore the

```



```

# line by adding a comment mark in front of it.
    stresults->addElement(list(spsns slot, spdts slot)); # saves the sampled values
    };
    getzoneMDread=ezn->zoneMD;
    zoneMDread=sampleNode(getzoneMDread);
    listres->addElement(list(zoneMDread, stresults)); # add the results of all starships for
each time step
    };
    puts("We had ",size(listres)," time steps. For each time step");
    puts(" 1 MD Reading and ", size(listres(0)(1))," Sensor Reports were created, thus we created\n");
    puts("a total of ",size(listres)," MD readings and ");
    puts(size(listres)*size(listres(0)(1))," Sensor Report results.\n");
    listres;
    };
# sampleNode(nd)
# nd -> node to be sampled
# samples the beliefs for a given node on the basis
# of its prior probabilities

def sampleNode(nd) {
    s = posteriorSamples(nd);
    smpl = s->nextSample();
    smpl(0)(1);
};

# printGTkb(gtkb);
# gtkb -> knowledge base made in makeGTkb();
# Prints the Ground Truth knowledge base, element by element

def printGTkb(gtkb) {
    icount=0; # counter for i
    for i in gtkb {
        puts("Results of time step ", icount, ":\n");
        puts("\tMD Reading result: ",i(0),".\n");
        puts("\tSensor Report results: \t");
        for j in i(1) {
            puts(j,"\t");
        };
        puts("\n");
        icount++;
    };
};

# cleanGTkb(gtkb)
# gtkb -> knowledge base made in makeGTkb();
# This function "cleans" the Ground Truth knowledge base by
# extracting the extra Sensor Reports. That is, the Sensor
# Reports that came out empty. This is the rationale:
# If at a given time step in the GTkb there is at least one
# positive result (i.e. either friend or enemy) then all "nothing"s
# are eliminated. Else, just one "nothing" result is kept.
# The idea is to represent what a real world sensor result would be.
# Since we do not know in advance how many starships are, we will
# only have a "nothing" as a result when the Sensors captured nothing
# in their range. If they captured only one, then this one is the only
# thing we know (i.e. there is no "nothing" result indicating that there is

```

```

# another starship but our sensor didn't catch it).

def cleanGTkb(gtkb){
    cleanedAll=array(0); #new, cleaned kb repository
    for i in gtkb{ #for each time step in the GTkb
        cleanedSR=array(0); #array for the cleaned Sensor Reports
        for j in i(1){ # for the list of Sensor Reports of that time step
            if !(j(0) == "nothing")
                then cleanedSR->addElement(j)
            end;
        };
        cleanedAll->addElement(list(i(0), cleanedSR));
    };
    cleanedAll;
};

# inferGT(clgtkb)
# cleanedgtkb -> knowledge base cleaned by cleanGTkb();
# Takes the results obtained from the starships in the
# Ground Truth knowledge base and tries to figure out
# how many starships are there and their respective type

## The functions below were built by Dr. Kathryn B. Laskey
# for the Display module of the Starship model.
#

puts("\nDefining the display functions\n");

def displayStatus(stpnum) {
    enqueueScript("displayStatusSub(" + stpnum + ")");
};

def displayStatusSub(stpnum) {
    for c in NamedEntity->retrieveInstances(false) {
        c->printDetail(); # display all the instances
    };
    $qv->updateDisplay();
    puts("Paused at " + stpnum + ". Hit <Enter> to continue...");
    while stdin->read() != 10 {}
};

puts("\nAll functions loaded!!\n");
#
# <<<<<<----- /STARSHIP_FUNCTIONS.SPI ----->>>>>>

```

### **Starship\_exec.spi**

```

#
#
# STARSHIP MODEL
#
# This file is part of the MEBN model inspired in the
# Paramount series Star Trek. The model was used in
# the PhD research of Paulo Costa and in the paper
# "MEBN without Multi-tears"
#
# Authors:

```

```

#           Paulo Cesar G da Costa
#           Kathryn B Laskey
#
# The model is composed of the following parts:
# Starship_main.spi - It's the execution manager for the Starship model
# Starship_framedefs.spi - Defines the model's frame structure
# Starship_functions.spi - Defines the functions used in the model
# Starship_ssbn.spi - Create instances and built an SSBN
#
# <<<<----- STARSHIP_EXEC.SPI ----->>>>
#
version_exec=v02; #defines which version of the model this file belongs to.
#
# This file defines the execution procedures that are used to build the SSBN
#
#           <<<<---CREATING ENTERPRISE AND ITS SPACE ZONE--->>>>
#
# We have to define in what type of space Enterprise
# will be flying. This is something we are supposed to
# know in advance (i.e., if our sensors are working, we
# know where we are).
#
puts("Creating Enterprise and its space zone...\n");
puts(".....\n");
#
area=Zone->makeInstance();
area->zoneNature= mainZone;
area->zoneEShips=numberToEShips(mainEShips);
area->zoneFShips=numberToFShips(mainFShips);

puts("Space area successfully defined...\n\n");
puts("We're now going to Create Enterprise...\n\n");

# Now, we create the Enterprise.
#
Enterprise=OwnStarship->makeInstance();
Enterprise->ownStarshipZone=area;                                # linking Enterprise with its zone

puts("Enterprise is ready for action...\n");
puts("As expected, we are navigating in a ",area->zoneNature->getValue()," zone.\n");
puts("Now our mission is to build the GTSet.\n");
#
#           <<<<---/CREATING ENTERPRISE AND ITS SPACE ZONE--->>>>
#
# <<<<--- BUILDING THE GROUND TRUTH KNOWLEDGE BASE --->>>>
#
puts("\nBuilding the Ground Truth Knowledge Base...\n\n" );

# Now we make a set of spaceships. This set can be
# used as Ground Truth for future experiments

GTSet=makeGTSet(area, mainEShips, mainCloakMode, mainFShips);
Enterprise->starship=GTSet; # assigns the set of starship instances to the Enterprise
area->starship=GTSet; # assigns the set of starship instances to the zone area

# Just to make sure everything is ok, let's print

```

```

# the initial characteristics of the starships

puts("\nAt this point we have created an initial set\n");
puts("of starships. \n");
puts("Let's see who they are and their characteristics:\n\n");
printCharSet(GTSet);

# For each of those starships, we will have a sensor report from
# the Enterprise sensor suite. Also, there will be one Magnetic
# Disturbance Report for each time step.
# We should build a Ground Truth database containing all the
# reports created during a given number of time steps on the
# basis of the starships inside the GTSet

puts(".....\n");
puts("Now, let's make the a database of reports that would\n");
puts("have been generatated by the Enterprise's sensors when\n");
puts("capturing data from those starships\n");
puts("Building the knowledge base.....\n");
puts(".....\n");

GTkb=buildGTkb(area, mainTimeSteps, GTSet);

puts("Knowledge base was built.....\n");
puts("Let's see the results:\n\n");
puts("Recall that we had the following starships in our ground truth:\n");
printCharSet(GTSet);
puts("Therefore, those where the starships that generated the \n");
puts("following reports:\n\n");
printGTkb(GTkb);
puts("\n\n.....\n");
#
# <<<<--- /BUILDING THE GROUND TRUTH KNOWLEDGE BASE --->>>>
#
# <<<<--- CLEANING THE GROUND TRUTH KNOWLEDGE BASE --->>>>
#
puts("\nCleaning the Ground Truth Knowledge Base...\n\n");

puts("Now, we are going to clean the knowledge base by extracting the\n");
puts("Sensor Reports that resulted in nothing, so there will be no clues on\n");
puts("whether we had stasheships with cloak devices around.\n");
puts("Here is the resulting cleaned GT knowledge base:\n\n");

cleanGT=cleanGTkb(GTkb);
printGTkb(cleanGT);

puts("\n\nTherefore, we know these results came from a Ground Truth set of ", size(GTSet), " starships \n");
puts("(", mainFShips, " friend(s) and ", mainEShips, " enemy(ies), where ", mainCloakMode);
puts(" enemy starship(s) had its Cloak Mode activated)\n");
puts("Now, we have to start from those reports");
puts(" and try to get as close as \npossible to the Ground Truth.\n\n");
#
# <<<<--- CLEANING THE GROUND TRUTH KNOWLEDGE BASE --->>>>
#
# <<<<----- STARSHIP_EXEC.SPI ----->>>>

```

## Appendix B Preliminary Syntax and Semantics for PR-OWL

### B.1 PR-OWL Classes

#### B.1.1 Alphabetical List of All PR-OWL Classes

The table below contains all classes used in the PR-OWL upper-ontology.

Table 6. Classes Used in PR-OWL

Class Name	Abbreviation	Sub-Classes (1 <sup>st</sup> level)
Argument Relationship	ArgRelationship	SimpleArgRelationship
Boolean Random Variable States	BooleanRVStates	
Built-In Random Variable	BuiltInRV	
Categorical Random Variable States	CategoricalRVStates	
Conditional Relationship	CondRelationship	
Context	Context	
Declarative Distribution	DeclarativeDist	
Domain MFrag	Domain_MFrag	
Domain Resident	Domain_res	
Entity	Entity	BooleanRVStates CategoricalRVStates MetaEntity ObjectEntity
Finding MFrag	Finding_MFrag	
Finding Resident	Finding_res	
Input	Input	Finding_input Generative_input
Meta-Entity	MetaEntity	
MFrag	MFrag	Domain_MFrag Finding_MFrag
MTheory	MTheory	

Node	Node	Context Input Resident
Object Entity	ObjectEntity	
Ordinary Variable	OVariable	
Probabilistic Assignment	ProbAssign	
Probabilistic Distribution	ProbDist	DeclarativeDist PR-OWLTable
PR-OWL Table	PR-OWLTable	
Resident	Resident	Finding_res Domain_res
Simple Argument Relationship	SimpleArgRelationship	
Skolem	Skolem	

### B.1.2 Detailed Explanation of PR-OWL Classes

#### **Argument Relationship (ArgRelationship)**

##### Description:

A generic random variable can have many arguments. Arguments are usually restricted in their type and meaning via the context nodes of an MFragment. In order to model these complex N-ary relations, PR-OWL makes use of the ArgRelationship class, which is a reified relation that conveys the number and order of arguments that each RV expects, its type (defined via a link to the OVariable class), and the link to the RV itself.

MEBN logic has the concept of a simple and a composite random variable term. Simple RV terms accept variables and constant symbols as arguments. Composite RV terms also accept other RV terms as arguments. In PR-OWL, the class ArgRelationship models composite RV terms, while its SimpleArgRelationship subclass models simple RV terms.

##### Subclasses:

SimpleArgRelationship

##### Properties with ArgRelationship as its domain (range inside parenthesis):

hasArgNumber (single xsd:nonNegativeInteger)

hasArgTerm (single Entity  $\sqcup$  OVariable  $\sqcup$  Resident)

isArgumentOf (single node)

### **Boolean Random Variable States (BooleanRVStates)**

#### Description

The BooleanRVStates class is formed by the Boolean truth-value states and the absurd symbol ( $\perp$ ). Individuals of this class are applied as possible values for Boolean random variables.

#### Subclasses:

None

#### Properties with BooleanRVStates as its domain (range inside parenthesis):

hasType (single MetaEntity)

hasUID (single xsd:string)

isPossibleValueOf (multiple Node  $\sqcup$  BuiltIRV)

### **Built-In Random Variable (BuiltInRV)**

#### Description:

Individuals of this class represent the random variables from MEBN logic's built-in MFrag: logical connectives, quantifiers, the equality random variable. Likewise their function in MEBN logic, these individuals allow PR-OWL ontologies to represent a rich family of probability distributions over interpretations of first-order logic.

Note that MEBN's built-in Indirect Reference MFrag is already represented in PR-OWL via its recursive scheme of building complex formulas shown in Chapter 5.

#### Subclasses:

None

#### Properties with BuiltInRV as its domain (range inside parenthesis):

hasContextInstance (multiple Context)

hasInputInstance (multiple Input)

hasPossibleValues (multiple Entity)

### **Categorical Random Variable States (CategoricalRVStates)**

#### Description:

Nodes represent random variables, which by definition have a list of mutually exclusive, collectively exhaustive states. In PR-OWL, those states are represented by individuals from class Entity. Some random variables have a list of categorical values as its possible states, and these are represented by elements from subclass CategoricalRVStates.

#### Subclasses:

None

#### Properties with CategoricalRVStates as its domain (range inside parenthesis):

hasType (single MetaEntity)

hasUID (single xsd:string)

isPossibleValueOf (multiple Node  $\sqcup$  BuiltInRV)

isArgTermIn (multiple ArgRelationship)

### **Conditional Relationship (CondRelationship)**

#### Description

The conditional relationship class is a reified property representing a (parent) node and one of its possible states. Individuals of this class are used to built 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 CondRelationship represents one parent/state pair, so a probability assignment is conditioned by a set of CondRelationship pairs (one for each parent node).

#### Subclasses:

None

#### Properties with CondRelationship as its domain (range inside parenthesis):

hasParentName (single Node)

hasParentState (single Entity)

isConditionantOf (multiple ProbAssign)



## **Context (Context)**

### Description

In general, MFragS impose constraints to the type of arguments each of its resident RVs should accept. The individuals of the Context class represent these types of constraints.

In PR-OWL, the class Context is the only subclass of the Node class that accepts composite RV terms as arguments (that is, uses the complete ArgRelationship instead of the more restricted SimpleArgRelationship).

A context node is either satisfiable or not, which means its possible states are instances of the BooleanRVStates class.

### Subclasses:

None

### Properties with Context as its domain (range inside parenthesis):

isContextInstanceOf (single Domain\_res  $\sqcup$  BuiltInRV)

isContextNodeIn (multiple Domain\_MFrag)

hasArgument (multiple ArgRelationship)

hasInnerTerm (multiple Node)

hasPossibleValues (multiple Entity)

isInnerTermOf (multiple Node)

isNodeFrom (multiple MFrag)

## **Declarative Distribution (DeclarativeDist)**

### Description

A declarative distribution is a distribution that is conveyed via a xsd:string datatype, using a specific format defined in the hasDeclaration datatype property. In order to allow a 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 proprietary format.

Describing a probability distribution is a much more compact and flexible way of conveying it. However, it assumes that an OWL-P parser 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 DeclarativeDist individuals for the same RV).

Subclasses:

None

Properties with DeclarativeDist as its domain (range inside parenthesis):

hasDeclaration (single xsd:string)

isRepresentedAs (single owl:oneOf{...list of possible formats...})

isProbDistOf (multiple resident)

isDefault (single xsd:Boolean)

### **Domain MFrag (Domain\_MFrag)**

Description

Domain MFrag is the subclass of class MFrag that includes all the domain-specific MFrag. It is disjoint with class Finding\_MFrag. All generative MFrag created by the ontology engineer (i.e. the domain expert) are members of this class.

Subclasses:

None

Properties with Domain\_MFrag as its domain (range inside parenthesis):

hasContextNode (multiple Context)

hasInputNode (multiple Input)

hasNode (multiple Node)

hasOVariable (multiple OVariable)

hasResidentNode (multiple Resident)

hasSkolem (multiple Skolem)

isMFragOf (multiple MTheory)

## **Domain Resident (Domain\_res)**

### Description

This is the subclass of class Resident (node) that includes all domain-specific resident nodes. It is disjoint with classes Finding\_res and BuiltInRV.

### Subclasses:

None

### Properties with Domain\_res as its domain (range inside parenthesis):

- hasContextInstance (multiple Context)
- hasArgument (multiple ArgRelationship)
- hasInnerTerm (multiple Node)
- hasParent (multiple resident  $\sqcup$  input)
- hasPossibleValues (multiple Entity)
- hasProbDist (multiple ProbDist)
- isArgTermIn (multiple ArgRelationship)
- isInnerTermOf (multiple Node)
- isNodeFrom (multiple MFrag)
- isParentOf (multiple resident)
- isResidentNodeIn (multiple MFrag)
- hasInputInstance (multiple Input)

## **Entity (Entity)**

### Description

MEBN logic treats the world as being comprised of entities that have attributes and are related to other entities. The logic assumes uniqueness of each concept (i.e. unique name assumption), so each entity in a MEBN model has a unique identifier and no unique identifier can be assigned to more than one entity.

PR-OWL follows MEBN syntax and semantics for defining entities so each member of the class Entity has a unique identifier assigned by the datatype property hasUID. OWL doesn't have the unique name assumption so the UID can be seen a tool for providing maximum compatibility with legacy OWL ontologies, since parsers may refer to it as a means to enforce uniqueness among declared entities.

It is important to note that not all concepts in an ontology have a UID, but only those which will be considered as part of the probabilistic model that is implicit in any probabilistic ontology. This structure allows mixing legacy deterministic ontologies with probabilistic ones. That allows knowledge engineers to assign PR-OWL definitions only to the parts of the domain for which plausible reasoning is desired.

Subclasses (1<sup>st</sup> level):

BooleanRVStates

CategoricalRVStates

MetaEntity

ObjectEntity

Properties with Entity as its domain (range inside parenthesis):

hasType (single MetaEntity)

hasUID (single xsd:string)

isArgTermIn (multiple ArgRelationship)

isPossibleValueOf (multiple Node  $\sqcup$  BuiltInRV)

**Finding MFrag (Finding MFrag)**

Description

Finding MFrag are used to convey information about findings, which is the default way of entering evidence in a MEBN MTheory so a probabilistic algorithm can be applied to perform inferences regarding the new evidence. They have no context nodes, only one input and one resident node.

Subclasses:

None

Properties with Finding\_MFrag as its domain (range inside parenthesis):

hasInputNode (single Input)

hasNode (multiple node)

hasOVariable (multiple OVariable)

hasResidentNode (multiple Resident)

hasSkolem (multiple Skolem)

isMFragOf (multiple MTheory)

### **Finding Resident (Finding\_res)**

#### Description

This is the subclass of class Resident (node) that includes all finding nodes. Finding nodes convey new evidence into a probabilistic system via a Finding\_MFrag. The class is disjoint with classes Domain\_res and BuiltInRV.

#### Subclasses:

None

#### Properties with Finding\_res as its domain (range inside parenthesis):

hasArgument (multiple ArgRelationship)  
 hasInnerTerm (multiple Node)  
 hasParent (single Finding\_input)  
 hasPossibleValues (multiple Entity)  
 hasProbDist (multiple ProbDist)  
 isArgTermIn (multiple ArgRelationship)  
 isInnerTermOf (multiple Node)  
 isNodeFrom (multiple MFrag)  
 isParentOf (multiple resident – cardinality = 0)  
 isResidentNodeIn (multiple MFrag)  
 hasInputInstance (multiple Input)

### **Input (Input)**

#### Description

In PR-OWL, an input node is basically a "copy" of a resident node that is used as an input in a given MFrag. Thus, each individual of class Input is linked with an individual of class Resident via the property isInputInstanceOf.

#### Subclasses:

Finding\_input  
 Generative\_input

Properties with Input as its domain (range inside parenthesis):

isInputInstanceOf (single Resident  $\sqcup$  BuiltInRV)  
 isInputNodeIn (multiple MFrag)  
 isParentOf (multiple Resident)  
 hasArgument (multiple ArgRelationship)  
 hasInnerTerm (multiple Node)  
 hasPossibleValues (multiple Entity)  
 isInnerTermOf (multiple Node)  
 isNodeFrom (multiple MFrag)

**Meta-Entity (MetaEntity)**Description

The MetaEntity class includes all the entities that convey specific definitions about entities (e.g. typelabels that name the possible types of entities).

Subclasses:

None

Properties with MetaEntity as its domain (range inside parenthesis):

isTypeOf (multiple Entity)  
 subsOVar (multiple OVariable)  
 hasType (single MetaEntity)  
 hasUID (single xsd:string)  
 isPossibleValueOf (multiple Node  $\sqcup$  BuiltInRV)  
 isArgTermIn (multiple ArgRelationship)

**MFrag (MFrag)**Description

MEBN Fragments (MFrag) are the basic structure of any MEBN logic model. MFrag represent influences among clusters of related RVs and can portray repeated patterns using ordinary variables as placeholders in to which entity identifiers can be substituted. In PR-OWL, each individual the MFrag class represents a MEBN Fragment (MFrag).

Subclasses:

Domain\_MFrag

Finding\_MFrag

Properties with MFrag as its domain (range inside parenthesis):

hasInputNode (multiple Input)

hasNode (multiple Node)

hasOVariable (multiple OVariable)

hasResidentNode (multiple Resident)

hasSkolem (multiple Skolem)

isMFragOf (multiple MTheory)

**MTheory (MTheory)**Description

An MTheory is a collection of MFrag that satisfies consistency constraints ensuring the existence of a unique joint distribution over the random variables mentioned in the MTheory.

In PR-OWL, the class MTheory allows a probabilistic ontology to have more than one valid MTheory to represent its RVs, and each individual of that class is basically a list of the MFrag that collectively form that MTheory. In addition, one MFrag can be part of more than one MTheory.

Subclasses:

None

Properties with MTheory as its domain (range inside parenthesis):

hasMFrag

**Node (Node)**Description

A node is part of an MFrag and it can be a random variable that is defined within that MFrag (a resident node), a RV that input values to nodes within that MFrag (an input node), or a RV that expresses the context in which the probability distributions within that MFrag are valid (a context node).

Subclasses (1<sup>st</sup> level):

Context

Input

Resident

Properties with Node as its domain (range inside parenthesis):

hasArgument (multiple ArgRelationship)

hasInnerTerm (multiple Node)

hasPossibleValues (multiple Entity)

isInnerTermOf (multiple Node)

isNodeFrom (multiple MFrag)

**Object Entity (ObjectEntity)**Description

The class ObjectEntity aggregates the MEBN entities that are real world concepts of interest in a domain. They are akin to objects in OO models and to frames in frame-based knowledge systems.

Subclasses:

None

Properties with ObjectEntity as its domain (range inside parenthesis):

hasType (single MetaEntity)

hasUID (single xsd:string)

isPossibleValueOf (multiple Node  $\sqcup$  BuiltInRV)

isArgTermIn (multiple ArgRelationship)

**Ordinary Variable (OVariable)**Description

Ordinary variables are placeholders used in MFrag's to refer to non-specific entities as arguments in a given MFrag's RVs.

Subclasses:



None

Properties with OVariable as its domain (range inside parenthesis):

isArgTermIn (multiple ArgRelationship)

isOVariableIn (single MFragment)

isRepBySkolem (multiple Skolem)

isSubsBy (single MetaEntity)

### **Probabilistic Assignment (ProbAssign)**

#### Description

Each cell in an PR-OWL table has a probability assignment for the state of a RV given the states of its parent nodes. Thus, the resulting relationship is N-ary and we opted for representing it via a reified relation (ProbAssign) that includes the name of the state to which the probability is being assigned, the probability value itself, and the list of states of parent nodes (i.e. conditionants) that collectively define the context in which that probability assignment is valid. Also, individuals of the ProbAssign class have an object property that links them with its respective PR-OWL table.

#### Subclasses:

None

Properties with ProbAssign as its domain (range inside parenthesis):

hasConditionant (multiple CondRelationship)

hasStateName (single Entity)

hasStateProb (single xsd:decimal)

isProbAssignIn (single PR-OWLTable)

### **Probabilistic Distribution (ProbDist)**

#### Description

This class is meant to represent the probability distributions that are defined in an MFragment to each of its resident nodes (random variables). A probability distribution can be described using a proprietary declarative format, such as a Netica table or a Quiddity function, or via a PR-OWL table (which has probability assignments as its cells).

#### Subclasses:

DeclarativeDist

PR-OWLTable

Properties with ProbDist as its domain (range inside parenthesis):

isDefault (single xsd:Boolean)

isProbDistOf (multiple Resident)

### **PR-OWL Table (PR-OWLTable)**

#### Description

An PR-OWL table has all the probability assignments for each state of a RV stored in a xsd:decimal format (future implementations might use the pr-owl:prob format, but currently that means incompatibilities with OWL, which has no support for PR-OWL custom datatypes).

This format for storing probability distributions cannot represent complex cases for which only formulas can represent a probability distribution (e.g. a node that have 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 ProbAssign class. Therefore, PR-OWL tables are only recommended for the simplest models in which the maximum level of compatibility is desired.

#### Subclasses:

None

Properties with PR-OWLTable as its domain (range inside parenthesis):

hasProbAssign (multiple ProbAssign)

isProbDistOf (multiple Resident)

isDefault (single xsd:Boolean)

### **Resident (Resident)**

#### Description

Resident nodes are the random variables that have their respective probability distribution defined in the MFRag.

#### Subclasses:

Domain\_res

Finding\_res

Properties with Resident as its domain (range inside parenthesis):

hasInputInstance (multiple Input)  
 hasParent (multiple Resident  $\sqcup$  Input)  
 hasProbDist (multiple ProbDist)  
 isArgTermIn (multiple ArgRelationship)  
 isParentOf (multiple Resident)  
 isResidentNodeIn (multiple MFrag)  
 hasArgument (multiple ArgRelationship)  
 hasInnerTerm (multiple Node)  
 hasPossibleValues (multiple Entity)  
 isInnerTermOf (multiple Node)  
 isNodeFrom (multiple MFrag)

**Simple Argument Relationship (SimpleArgRelationship)**Description

Each generic random variable can have many arguments. Arguments are usually restricted in their type and meaning via the context nodes of an MFrag. In order to model these complex N-ary relations, PR-OWL makes use of the SimpleArgRelationship class, which is a reified relation that conveys the number and order of arguments that each RV expects, it's type (defined via a link to the TypeContext class), and the link to the RV itself.

Subclasses:

None

Properties with SimpleArgRelationship as its domain (range inside parenthesis):

hasArgNumber (single xsd:nonNegativeInteger)  
 hasArgTerm (single Entity  $\sqcup$  OVariable  $\sqcup$  Resident  $\sqcup$  Skolem)  
 isArgumentOf (single Node)

**Skolem (Skolem)**Description

Each individual of class Skolem represents a Skolem constant in a MEBN quantifier random variable. Each MEBN quantifier random variable corresponds to a first-order formula beginning with a universal or existential quantifier. The Skolem constant in the MEBN random variable represents a generic individual within the scope of the universal or existential quantifier of the corresponding first-order formula.

MEBN logic contains a set of built-in MFragS for quantifier random variables. In PR-OWL modelers can use individuals of class Skolem to define distributions for Skolem constants used in quantifier random variables.

Subclasses:

None

Properties with Skolem as its domain (range inside parenthesis):

isArgTermIn (multiple ArgRelationship)

isSkolemIn (multiple MFrag)

representsOVar (single OVariable)

## B.2 PR-OWL Properties

### B.2.1 Alphabetical List of All PR-OWL Properties

The table below contains all properties used in the PR-OWL upper-ontology.

Table 7. Properties Used in PR-OWL

Property Name	Domain	Range	Inverse Property
hasArgNumber	ArgRelationship	xsd:nonNegativeInteger	-x-
hasArgTerm	ArgRelationship	Entity OVariable Resident Skolem	isArgTermIn
hasArgument	Node	ArgRelationship	isArgumentOf
hasConditionant	ProbAssign	CondRelationship	isConditionantOf
hasContext	Domain_MFrag	Context	isContextIn
hasContextInstance	Domain BuiltInRV	Context	isContextInstanceOf
hasDeclaration	DeclarativeDist	xsd:string	-x-

hasInnerTerm	Node	Node	isInnerTermOf
hasInputInstance	Resident BuiltInRV	Input	isInputInstanceOf
hasInputNode	MFrag	Input	isInputNodeIn
hasMFrag	MTheory	MFrag	isMFragOf
hasOVariable	MFrag	OVariable	isOVariableIn
hasParent	Resident	Resident Input	isParentOf
hasParentName	CondRelationship	Node	
hasParentState	CondRelationship	Entity	
hasPossibleValues	BuiltInRV Node	Entity	isPossibleValueOf
hasProbAssign	PR-OWLTable	ProbAssign	isProbAssignIn
hasProbDist	Resident	ProbDist	isProbDistOf
hasResidentNode	MFrag	Resident	isResidentNodeIn
hasSkolem	MFrag	Skolem	isSkolemIn
hasStateName	ProbAssign	Entity	
hasStateProb	ProbAssign	xsd:decimal	
hasType	Entity	MetaEntity	isTypeOf
hasUID	Entity	xsd:string	
isArgTermIn	OVariable Resident Entity Skolem	ArgRelationship	hasArgTerm
isArgumentOf	ArgRelationship	Node	hasArgument
isConditionantOf	CondRelationship	ProbAssign	hasConditionant
isContextIn	Context	Domain_MFrag	hasContext
isContextInstanceOf	Context	Domain BuiltInRV	hasContextInstance
isDefault	ProbDist	xsd:boolean	
isInnerTermOf	Node	Node	hasInnerTerm
isInputInstanceOf	Input	Resident BuiltInRV	hasInputInstance
isInputNodeIn	Input	MFrag	hasInputNode
isMFragOf	MFrag	MTheory	hasMFrag
isOVariableIn	OVariable	MFrag	hasOVariable
isParentOf	Resident Input	Resident	hasParent

isPossibleValueOf	Entity	Node BuiltInRV	hasPossibleValues
isProbAssignIn	ProbAssign	PR-OWLTable	hasProbAssign
isProbDistOf	ProbDist	Resident	hasProbDist
isRepBySkolem	OVariable	Skolem	representsOVar
isRepresentedAs	DeclarativeDist	Owl:one of{...}	
isResidentNodeIn	Resident	MFrag	hasResidentNode
isSkolemIn	Skolem	MFrag	hasSkolem
isSubsBy	OVariable	MetaEntity	
isTypeOf	MetaEntity	Entity	hasType
representsOVar	Skolem	OVariable	isRepBySkolem
subsOVar	MetaEntity	OVariable	isSubsBy

### B.2.2 Detailed Explanation of PR-OWL Properties

#### **hasArgNumber**

Type: Datatype property

Description:

This datatype property assigns the argument number of an argument relationship. As an example, if we have a random variable with 3 arguments, it will have three ArgRelationship reified relations. The first argument of the RV will have the number 1 assigned to its respective hasArgNumber property, the second will have the number 2 assigned and the third will have the number 3 assigned. In short this property keeps track of the ordering between the arguments of an RV.

The datatype range is a nonNegativeInteger. We used this instead of a positiveInteger because we wanted zero as a possible value, since we assume that a RV with no arguments means a global RV.

#### **hasArgTerm**

Type: Object property

Description:

This object property links one instance of class ArgRelationship (which is linked to a RV) to an internal variable within the home MFrag where its RV is resident, to a node that is being used as argument in that RV, or to a MEBN entity.

One individual of the class ArgRelationship can have only one RV (since it refers to a specific argument of an RV), and thus can be related to only one OVariable (Simple RV Terms) or one Node (Composite RV Terms), which makes that property a functional one.

The inverse property is isArgTermIn.

### **hasArgument**

Type: Object property

Description:

This object property is the link between a node in an MFrag and the reified relation that conveys its respective arguments. Note that each instance of a node will have only one argument relationship, which is defined within that node's MFrag.

The inverse of this property is isArgumentOf.

### **hasConditionant**

Type: Object property

Description:

Each instance of the class ProbAssign corresponds to the probability assignment for a given state of a RV. This probability assignment is conditioned by the parent RVs of that RV. This object property conveys the list of the states of the parent RV which have influenced that specific probability assignment. Since any MEBN entity can be a state in a RV, this property has MEBNEntity class as its range.

The inverse property is isConditionantOf.

### **hasContextInstance**

Type: Object property

Description:

This object property links a resident node or a built-in RV to its many possible "context node instances", or the instances of context nodes that take their values from that resident node or built-in RV.

The inverse property is isContextInstanceOf.

### **hasContextNode**

Type: Object property

Description:

This object property links an MFrag to the context nodes being applied to it.

The inverse property is isContextIn.

### **hasDeclaration**

Type: Datatype property

Description:

This datatype property conveys the declarative probability distributions. Each probability distribution can be expressed in different formats and each format is defined by the datatype property isRepresentedAs. Possible formats include Netica tables, Netica equations, Quiddity formulas, MEBN syntax, and others. However, the declaration itself is stored as a string so parsers are expected to understand how to deal with the specific text format of each declaration.

### **hasInnerTerm**

Type: Object property

Description:

This object property makes the connection between the many possible inner terms inside a MENB equation. It is used to decompose random variable terms usually employed in context and input nodes.

The inverse property is isInnerTermOf

### **hasInputInstance**

Type: Object property

Description:

This object property links a resident node or a built-in RV to its many possible "input node instances", or the instances of input nodes that take their values from that resident node or built-in RV.



The inverse property is `isInputInstanceOf`.

### **hasInputNode**

Type: Object property

Description:

This object property links each MFrag with its respective input nodes.

The inverse property is `isInputNodeIn`.

### **hasMFrag**

Type: Object property

Description:

This object property links one MTheory with its respective MFrag. Usually, a probabilistic ontology will have only one MTheory as a means to convey the global joint probability distribution of its random variables. However, MEBN logic allows many possible MTheories to represent a given domain, so it is reasonable to infer that in some circumstances it might be preferable to have one probability ontology being represented by more than one MTheory.

The inverse property is `isMFragOf`.

### **hasNode**

Type: Object property

Description:

This object property links one MFrag with its respective nodes.

The inverse property is `isNodeFrom`.

### **hasOVariable**

Type: Object property

Description:

This inverse functional object property relates one MFrag to its ordinary variables (i.e. individuals from class OVariable that are related to the MFrag).

The inverse of this property is `isOVariableIn`.

**hasParent**

Type: Object property

Description:

This object property links a resident node of an MFragment with its respective parent(s), which has(have) to be an individual of either the class Resident or the class Input.

The inverse property is isParentOf.

**hasParentName**

Type: Object property

Description:

This object property links a CondRelationship to a Node. The reified conditional relationship is used to build PR-OWL Tables. One table usually has many probability assignments (which correspond to cells in a table), and each probability assignment has a set of conditionants. Conditionants are the states of the parents of a node that form a combination where a given probability assignment holds. Each CondRelationship defines a pair parent/state-of-parent, and the hasParentName property defines the parent name of that pair.

**hasParentState**

Type: Object property

Description:

This object property links a CondRelationship to an Entity. The reified conditional relationship is used to build PR-OWL Tables. One table usually has many probability assignments (which correspond to cells in a table), and each probability assignment has a set of conditionants. Conditionants are the states of the parents of a node that form a combination where a given probability assignment holds. Each CondRelationship defines a pair parent/state-of-parent, and the hasParentState property defines the parent state of that pair.

**hasPossibleValues**

Type: Object property

Description:

This object property defines what are the possible values of a node in an MFragment (which is by definition a random variable). Possible states include all kinds of entities.

The inverse property is `isPossibleValueOf`.

### **hasProbAssign**

Type: Object property

Description:

A PR-OWL table is formed by many individual members of the class `ProbAssign`, which are cells in that table. This object property relates one PR-OWL table to its respective cells (`ProbAssign` elements).

The inverse property is `isProbAssignIn`.

### **hasProbDist**

Type: Object property

Description:

This object property links a RV to its respective probability distributions, as defined in that RV's home MFrags. Note that this property is not being defined as functional, implying a polymorphic version of MEBN (where each RV can have different distributions in different MFrags).

The inverse of this property is `isProbDistOf`.

### **hasResidentNode**

Type: Object property

Description:

This object property links an MFragment with its respective resident node(s).

The inverse property is `isResidentNodeIn`

### **hasSkolem**

Type: Object property

Description:

This object property relates one MFrag with the Skolem constants (i.e. an individual from class Skolem) that are defined in that MFrag.

The inverse of this property is isSkolemIn.

### **hasStateName**

Type: Object property

Description:

When a probability distribution is conveyed as an PR-OWL table, each individual cell is represented as an individual of the ProbAssign class. This object property refers to which state of a random variable (i.e. MFrag node) a given probability assignment refers.

The property itself is functional, since one state can have only one probability assignment for the configuration listed in each individual of the ProbAssign class.

### **hasStateProb**

Type: Datatype property

Description:

This datatype property is used to store the actual probability of an individual ProbAssign. Currently, OWL has no support for user defined datatypes, so instead of using owl-p:prob datatype (which includes all decimals between 0 and 1 inclusive) we are using xsd:decimal for compatibility purposes.

### **hasType**

Type: Object property

Description:

In the extended MEBN logic that is the backbone of PR-OWL, each and every entity has a type. The list of types consists of the individuals from class MetaEntity. This functional object property defines the type of each entity by linking it to an individual of the MetaEntity class.

Every entity has a MetaEntity (TypeLabel, CategoryLabel, Boolean, or a domain-specific label) as a Type. As an example, an hypothetical individual of an ObjectEntity class named Starship would have type Starship, which is a domain-specific label for an ObjectEntity individual that happens to be a starship. That domain-specific label is itself an individual of the MetaEntity class.

The inverse property is isTypeOf.

### **hasUID**

Type: Datatype property

Description:

MEBN logic has the unique naming assumption, which is not assumed in OWL (even though tools such as Protégé make that assumption for improved reasoning purposes). In order to make sure that a tool that does not assume unique identifiers would not prevent MEBN reasoners to work, each MEBN entity has a unique identifier assigned by this datatype property.

The UID itself is conveyed as a xsd:string, and the hasUID datatype property is declared as functional in order to enforce uniqueness.

### **isArgTermIn**

Type: Object property

Description:

This object property links an individual of class OVariable, Resident, Entity, or Skolem to one ArgRelationship(s) that has individual as its argument. Each ArgRelationship can have only one argument, but each individual of those classes can refer to many ArgRelationships.

The inverse of this property is hasArgTerm.

### **isArgumentOf**

Type: Object property

Description:

This object property links an Argument Relationship to its respective Node (i.e. to the individual of class Node that has this ArgRelationship into its argument list).

The inverse of this property is hasArgument.

### **isConditionantOf**

Type: Object property

Description:

This object property links one possible state of a parent node to the configuration that is conditioning its children state's probability distribution.

The inverse property is hasConditionant.

**isContextInstanceOf**

Type: Object property

Description:

This object property links a context node to its respective "generative resident node" or built-in RV (i.e. the resident node or built-in RV from which the context node is a pointer).

The inverse property is hasContextInstance.

**isContextNodeIn**

Type: Object property

Description:

This object property links one context node to the respective MFrag in which that context node applies.

The inverse property is hasContext.

**isDefault**

Type: Datatype property

Description:

This datatype property indicates whether a probability distribution is the default probability distribution of a node or not. Default probability distributions for nodes are used when the context nodes of the MFrag containing those nodes are not met.

**isInputInstanceOf**

Type: Object property

Description:

This object property links an input node to its "generative resident node", or the resident node to which that input node is a copy.

The inverse property is `hasInputInstance`.

### **isInputNodeIn**

Type: Object property

Description:

This object property links a node to the MFrag that have it as an input.

The inverse property is `hasInputNode`

### **isMFragOf**

Type: Object property

Description:

This object property links one MFrag to one or more MTheories (i.e. individuals of class MTheory) that have that MFrag as its component.

The inverse property is `hasMFrag`.

### **isNodeFrom**

Type: Object property

Description:

This general object property links one node to the MFrag it belongs to.

The inverse property is `hasNode`.

### **isOVariableIn**

Type: Object property

Description:

This functional object property relates one ordinary variable (i.e. an individual from class OVariable) to its respective MFrag.

The inverse of this property is `hasOVariable`.

**isParentOf**

Type: Object property

Description:

This object property links a resident or input node of an MFragment with its respective children, which are resident nodes in that same MFragment.

The inverse property is hasParent.

**isPossibleValueOf**

Type: Object property

Description:

This object property correlates one entity with the node(s) of one or more MFrags that have such entity as a possible state.

Note that the individuals listed as being possible values of a node must form a mutually exclusive, collectively exhaustive set. PR-OWL has the same tools for enforcing exclusiveness (i.e. MEBN entity unique name assumption and the existential and universal qualifiers acting together as a closure axiom), but the domain expert must ensure completeness.

The inverse property is hasPossibleValues.

**isProbAssingIn**

Type: Object property

Description:

This is the inverse of the hasProbAssign object property and links one individual probability assignment to its respective probability distribution table.

**isProbDistOf**

Type: Object property

Description:

This object property links a probability distribution to its respective RV (resident node). Note that this property is functional, since each probability distribution in a MFragment defines a unique RV.



The inverse of this property is hasProbDist.

### **isRepBySkolem**

Type: Object property

Description:

This object property links one ordinary variable to the Skolem constant that represents that ordinary variable in quantified expressions. The property is inverse functional, since one Skolem constant can represent only the group of entities that can be replaced with that ordinary variable in the model.

The inverse property is representsOVar.

### **isRepresentedAs**

Type: Datatype property

Description:

This datatype property defines how a given declarative probability distribution is expressed. Each probability distribution can be expressed in different formats, and each format is defined by this datatype property. Possible formats include Netica tables, Netica equations, Quiddity formulas, MEBN syntax, and others. However, the declaration itself is stored in the hasDeclaration datatype property as a string so parsers will have to know how to deal with the specific text format of each declaration.

### **isResidentNodeIn**

Type: Object property

Description:

This object property links an individual of class Node to the MFrag(s) that have this node as a resident node.

The inverse property is hasResidentNode.

### **isSkolemIn**

Type: Object property

Description:

This object property relates one Skolem constant (i.e. an individual from class Skolem) to the MFragment in which it is defined.

The inverse of this property is hasSkolem.

### **isSubsBy**

Type: Object property

Description:

This object property links one instance of class OVariable to type of the entity that can substitute it. Each argument of a RV has its expected type defined within the home MFragment of that RV. In PR-OWL, the type restrictions are defined directly through the OVariable using the isSubsBy property. One MFragment can have many OVariables (which can be themselves linked to many SimpleArgRelationships) but each OVariable has a unique type, which is explicitly defined by the type of the entity that can substitute that OVariable.

This object property is the inverse of subsOVar.

### **isTypeOf**

Type: Object property

Description:

This is the inverse of hasType object property, and basically lists all the MEBN entities that have its respective type defined by that specific individual of either the MetaEntity class or the ObjectEntity class.

### **representsOVar**

Type: Object property

Description:

This object property links a Skolem constant (i.e. an individual of class Skolem) to the ordinary variable it represents in a quantifier expression. The property is functional since each Skolem constant represents only one ordinary variable in the model.

The inverse property is isRepBySkolem.

### **subsOVar**

Type: Object property

Description:

This object property assigns MetaEntity individuals in order to define the type of the substituters for each MFrag ordinary variable.

Its inverse property is the functional isSubsBy.

### B.3 Naming Convention (optional)

For naming purposes, PR-OWL elements can be partitioned in two major groups: Entities and all the other elements. This distinction comes from the fact that the second group is basically a set of classes that provide the support for the probabilistic part of an ontology. That is, the second group is the backbone of the MEBN-based probabilistic representation that collectively form the PR-OWL semantics. As a result, all of that supporting set elements are linked to an MFrag, which is the basic structure of a MEBN model, and a great level of consistency and straightforwardness can be achieved by adopting a naming convention that acknowledges this fact.

Therefore, a very simple, optional naming convention was used in this research and has proved to be an important asset for keeping consistency and making maintenance of the model easier. In addition, future implementations geared to facilitate the creation / edition of probabilistic ontologies would certainly keep the majority of those supporting elements hidden from the normal user so it seems reasonable that such a system performs an automatic naming for those hidden elements.

#### **Non-Entity elements**

The convention adopted was based on blocks divided by underscore characters, while multiple words within a block should be separated using the “camelback” notation (e.g. KeepingTheFirstLetterCapitalized). Also, an ordering among the blocks should be followed to enable any reader aware of the notation to infer the meaning of each element on the basis of its name only. The general format is:

First block - MFrag: First letter(s) of the MFrag to which the element is linked. If there are two MFrag with the same first letter then subsequent letters of the name should be used (non-capitalized) until the ambiguity is resolved.

Second block – Name/relationship: name of the element or of its related node. Usually, names of OVariables and Nodes are not abbreviated, while the longer names of context or input RV terms should have most of its elements abbreviated.

Third block – Type (optional): type of an element or of its related node. This is an optional block that has only one, non-abbreviated and non-capitalized word. Standard types: context, input, ddecl (default declarative distribution), decl (non-default declarative

distribution), dtable (PR-OWL Table default distribution), table (PR-OWL Table non-default distribution), cond (conditionant).

Fourth block – Discriminator (optional): This block should be used to discriminate similar elements. When more than one number is used, separation is made using a dot (e.g. 2.4 meaning the second of four elements in an argument relationship or the second ).

Here are some examples and their respective intended meaning:

**SRD\_sr:** Ordinary variable “sr” from the Sensor Report Data MFrags.

**S\_CloakMode:** Resident node “Cloak Mode” from the Starship MFrags (if an input or context node then a type block would be necessary).

**Z\_CloakMode\_input:** Input node “Cloak Mode” from the Zone MFrags.

**Z\_ZoneEShips\_ddecl\_Netica:** Default declarative distribution of node ZoneEShips from Zone MFrags, written in Netica format.

**Z\_TprevPrevT\_context:** Context node “(tprev = Prev(t))” from the Zone MFrags.

**Z\_TprevPrevT\_inner\_prevT:** Inner term “Prev(t)” of context node “(tprev = Prev(t))” from the Zone MFrags.

**Z\_TprevPrevT\_inner\_prevT\_2.2:** Second argument (out of 2) from the argument relationship (ArgRelationship) of the inner term “Prev(t)” of context node “(tprev = Prev(t))” from the Zone MFrags.

**Z\_ZoneEShips\_table\_4.3.5:** Probabilistic assignment for the fourth state of the variable ZoneEShips from the Zone MFrags, given the third state of one of its parents and the fifth state of its other parent (both states are represented as CondRelationships, which include the name of the parent and its respective state).

**DTS\_OpSpec\_inputCond\_2.3:** Conditionant relationship representing the second state out of three states of input node OpSpec from the DangerToSelf MFrags. Note that the type block has two values (input and cond) so the “camelback” notation is used to separate those values inside the same block

Exceptions:

MFrags – Names of MFrags will be stated in the first block (not abbreviated) followed by the suffix “\_MFrags”. Example: **Starship\_MFrags, DangerToSelf\_MFrags**.

MTheorys – Names of MTheories will be stated in the first block (not abbreviated) followed by the suffix “\_MTheory”. Example: **StarTrek\_MTheory, Confederation\_MTheory**.

Built-In RVs – PR-OWL built-in RVs cannot be changed by the probabilistic ontology editor and should be used as is.

### **Entity elements**

Each of the four types of entity element has its own peculiarity:

**Boolean RV States**: These are built-in to PR-OWL, so cannot be changed.

**Categorical RV States**: Names of categorical states should be preceded by a block containing the first letters (capitalized) of the RV (node) they are state from. If there are two RVs with the same first letter then subsequent letters of the name should be used (non-capitalized) until the ambiguity is resolved. The name of the state itself is up to the ontology engineer; provide that the unique naming assumption is respected.

**Meta Entities**: Built-In Meta Entities cannot be changed. Domain-specific Meta Entities should have the same name of their respective object class followed by the suffix “\_Label”. Example: the Meta Entity that designate the type of individuals of class Starship should be named Starship\_Label.

**Object Entities**: The ontology engineer is free to choose any naming for the classes of object entities and for its respective individuals, provide that the unique naming assumption is respected.

## **B.4 PR-OWL Upper-Ontology Code**

The following OWL code includes all the elements of the PR-OWL extension formatted as an upper ontology.

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:owl="http://www.w3.org/2002/07/owl#"
  xmlns="http://mason.gmu.edu/~pcosta/pr-owl/pr-owl.owl#"
  xml:base="http://mason.gmu.edu/~pcosta/pr-owl/pr-owl.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="OVariable">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >Ordinary variables are placeholders used in MFrag's to refer to non-specific entities as arguments in a
      given MFrag's RVs.</rdfs:comment>
    <owl:equivalentClass>
      <owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
          <owl:Restriction>
            <owl:allValuesFrom>
              <owl:Class rdf:ID="ArgRelationship"/>
            </owl:allValuesFrom>
            <owl:onProperty>
              <owl:ObjectProperty rdf:ID="isArgTermIn"/>
            </owl:onProperty>
          </owl:Restriction>
        </owl:intersectionOf>
      </owl:Class>
    </owl:equivalentClass>
  </owl:Class>
```

```

<owl:onProperty>
  <owl:ObjectProperty rdf:about="#isArgTermIn"/>
</owl:onProperty>
<owl:someValuesFrom>
  <owl:Class rdf:about="#ArgRelationship"/>
</owl:someValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty>
    <owl:ObjectProperty rdf:ID="isOVariableIn"/>
  </owl:onProperty>
  <owl:allValuesFrom>
    <owl:Class rdf:ID="MFrag"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:someValuesFrom>
    <owl:Class rdf:about="#MFrag"/>
  </owl:someValuesFrom>
  <owl:onProperty>
    <owl:ObjectProperty rdf:about="#isOVariableIn"/>
  </owl:onProperty>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty>
    <owl:ObjectProperty rdf:about="#isOVariableIn"/>
  </owl:onProperty>
  <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:cardinality>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty>
    <owl:FunctionalProperty rdf:ID="isSubsBy"/>
  </owl:onProperty>
  <owl:allValuesFrom>
    <owl:Class rdf:ID="MetaEntity"/>
  </owl:allValuesFrom>
</owl:Restriction>
<owl:Restriction>
  <owl:someValuesFrom>
    <owl:Class rdf:about="#MetaEntity"/>
  </owl:someValuesFrom>
  <owl:onProperty>
    <owl:FunctionalProperty rdf:about="#isSubsBy"/>
  </owl:onProperty>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty>
    <owl:FunctionalProperty rdf:about="#isSubsBy"/>
  </owl:onProperty>
  <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:cardinality>
</owl:Restriction>
<owl:Restriction>
  <owl:onProperty>
    <owl:ObjectProperty rdf:ID="representsOVar"/>

```

```

    </owl:onProperty>
    <owl:allValuesFrom>
      <owl:Class rdf:ID="Skolem"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="BuiltInRV">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Individuals of this class represent the random variables from MEBN logic's built-in MFragments: logical
    connectives, quantifiers, the equality random variable. Likewise their function in MEBN logic, these
    individuals allow PR-OWL ontologies to represent a rich family of probability distributions over interpretations
    of first-order logic.
    Note that MEBN's built-in Indirect Reference MFragment is already represented in PR-OWL via its recursive
    scheme of building complex formulas.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasPossibleValues"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="BooleanRVStates"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:ID="Input"/>
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasInputInstance"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:ID="Domain_Res"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#BooleanRVStates"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:ID="Finding_res"/>
  </owl:disjointWith>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:ID="Context">

```

```

<rdfs:subClassOf>
  <owl:Class rdf:ID="Node"/>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="hasInnerTerm"/>
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#Context"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="isContextInstanceOf"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
    </owl:onProperty>
    <owl:someValuesFrom>
      <owl:Class rdf:ID="Entity"/>
    </owl:someValuesFrom>
  </owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith>
  <owl:Class rdf:ID="Resident"/>
</owl:disjointWith>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
    </owl:onProperty>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#Entity"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >In general, MFragments impose constraints to the type of arguments each of its resident RVs should accept.
  The individuals of the Context class represent these types of constraints.
  In PR-OWL, the class Context is the only subclass of the Node class that accepts composite RV terms as
  arguments (that is, uses the complete ArgRelationship instead of the more restricted
  SimpleArgRelationship).
  A context node is either satisfiable or not, which means its possible states are instances of the
  BooleanRVStates class.
</rdfs:comment>
<owl:disjointWith>
  <owl:Class rdf:about="#Input"/>
</owl:disjointWith>
<rdfs:subClassOf>

```



```

<owl:Restriction>
  <owl:allValuesFrom>
    <owl:Class rdf:ID="Domain_MFrag"/>
  </owl:allValuesFrom>
  <owl:onProperty>
    <owl:ObjectProperty rdf:ID="isContextNodeIn"/>
  </owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Domain_MFrag">
  <owl:disjointWith>
    <owl:Class rdf:ID="Finding_MFrag"/>
  </owl:disjointWith>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Domain MFrag is the subclass of class MFrag that includes all the domain-specific MFrag. It is disjoint
    with class Finding_MFrag. All generative MFrag created by the ontology engineer (i.e. the domain expert)
    are members of this class.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#MFrag"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom rdf:resource="#Context"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasContextNode"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="CondRelationship">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="hasParentState"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Entity"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasParentName"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#hasParentState"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>

```

```

    <owl:allValuesFrom>
      <owl:Class rdf:about="#Entity"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasParentName"/>
    </owl:onProperty>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#Node"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >The conditional relationship class is a reified property representing a (parent) node and one of its
  possible states. Individuals of this class are used to built 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 CondRelationship represents one parent/state pair, so a
  probability assignment is conditioned by a set of CondRelationship pairs (one for each parent
  node).</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Node"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasParentName"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#hasParentState"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="ProbAssign"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isConditionantOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ProbDist">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>

```

```

    <owl:ObjectProperty rdf:ID="isProbDistOf"/>
  </owl:onProperty>
  <owl:someValuesFrom>
    <owl:Class rdf:about="#Resident"/>
  </owl:someValuesFrom>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>This class is meant to represent the probability distributions that are defined in an MFragment to each of its
resident nodes (random variables). A probability distribution can be described using a proprietary declarative
format, such as a Netica table or a Quiddity function, or via an PR-OWL table (which has probability
assignments as its cells).</rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#Resident"/>
    </owl:allValuesFrom>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isProbDistOf"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="PR-OWLTable">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Resident"/>
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isProbDistOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="#ProbDist"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#ProbAssign"/>
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:InverseFunctionalProperty rdf:ID="hasProbAssign"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>An PR-OWL table has all the probability assignments for each state of a RV stored in a xsd:decimal
format (future implementations might use the pr-owl:prob format, but currently that means incompatibilities
with OWL, which has no support for PR-OWL custom datatypes).
This format for storing probability distributions cannot represent complex cases for which only formulas can
represent a probability distribution (e.g. a node that have 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 ProbAssign
class. Therefore, PR-OWL tables are only recommended for the simplest models in which the maximum
level of compatibility is desired.</rdfs:comment>
  <owl:disjointWith>

```

```

    <owl:Class rdf:ID="DeclarativeDist"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:InverseFunctionalProperty rdf:about="#hasProbAssign"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#ProbAssign"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isProbDistOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Resident"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isProbDistOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Finding_MFrag">
  <owl:disjointWith rdf:resource="#Domain_MFrag"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:ID="Finding_input"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasInputNode"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Finding MFragments are used to convey information about findings, which is the default way of entering
    evidence in a MEBN MTheory so a probabilistic algorithm can be applied to perform inferences regarding
    the new evidence. They have no context nodes, only one input and one resident node. </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasResidentNode"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Finding_res"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>

```

```

    </owl:Restriction>
  </rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasResidentNode"/>
    </owl:onProperty>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasInputNode"/>
    </owl:onProperty>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#Finding_input"/>
    </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:someValuesFrom>
      <owl:Class rdf:about="#Finding_res"/>
    </owl:someValuesFrom>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasResidentNode"/>
    </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasInputNode"/>
    </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Class rdf:about="#MFrag"/>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ObjectEntity">
  <owl:disjointWith>
    <owl:Class rdf:about="#BooleanRVStates"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Entity"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom rdf:resource="#OVariable"/>
      <owl:onProperty>
        <owl:InverseFunctionalProperty rdf:ID="subsOVar"/>

```

```

    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith>
  <owl:Class rdf:ID="CategoricalRVStates"/>
</owl:disjointWith>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>The class ObjectEntity aggregates the MEBN entities that are real world concepts of interest in a
domain. They are akin to objects in OO models and to frames in frame-based knowledge
systems.</rdfs:comment>
  <owl:disjointWith>
    <owl:Class rdf:about="#MetaEntity"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#Input">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#Resident"/>
      </owl:someValuesFrom>
    </owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="isParentOf"/>
    </owl:onProperty>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#ArgRelationship"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#ArgRelationship"/>
      </owl:allValuesFrom>
    </owl:Restriction>
    <owl:onProperty>
      <owl:InverseFunctionalProperty rdf:ID="hasArgument"/>
    </owl:onProperty>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#MFRag"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isInputNodeIn"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#MFRag"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isInputNodeIn"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#MFRag"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</rdfs:subClassOf>

```

```

<owl:Restriction>
  <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:cardinality>
  <owl:onProperty>
    <owl:ObjectProperty rdf:ID="isInputInstanceOf"/>
  </owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasInnerTerm"/>
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#Input"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isParentOf"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >In PR-OWL, an input node is basically a "copy" of a resident node that is used as an input in a given
  MFrag. Thus, each individual of class Input is linked with an individual of class Resident via the property
  isInputInstanceOf.</rdfs:comment>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isInputInstanceOf"/>
    </owl:onProperty>
    <owl:someValuesFrom>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Resident"/>
          <owl:Class rdf:about="#BuiltInRV"/>
        </owl:unionOf>
      </owl:Class>
    </owl:someValuesFrom>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Class rdf:about="#Node"/>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#Context"/>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isInputInstanceOf"/>
    </owl:onProperty>
    <owl:allValuesFrom>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">

```

```

        <owl:Class rdf:about="#Resident"/>
        <owl:Class rdf:about="#BuiltInRV"/>
    </owl:unionOf>
</owl:Class>
</owl:allValuesFrom>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith>
    <owl:Class rdf:about="#Resident"/>
</owl:disjointWith>
<rdfs:subClassOf>
    <owl:Restriction>
        <owl:allValuesFrom>
            <owl:Class rdf:about="#Resident"/>
        </owl:allValuesFrom>
    </owl:onProperty>
        <owl:ObjectProperty rdf:about="#isParentOf"/>
    </owl:onProperty>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Finding_res">
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <owl:ObjectProperty rdf:ID="hasParent"/>
            </owl:onProperty>
            <owl:someValuesFrom>
                <owl:Class rdf:about="#Finding_input"/>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty>
                <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
            </owl:onProperty>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:oneOf rdf:parseType="Collection">
                        <BooleanRVStates rdf:ID="absurd">
                            <isPossibleValueOf>
                                <BuiltInRV rdf:ID="equalto">
                                    <hasPossibleValues>
                                        <BooleanRVStates rdf:ID="true">
                                            <isPossibleValueOf>
                                                <BuiltInRV rdf:ID="or">
                                                    <hasPossibleValues rdf:resource="#true"/>
                                                <hasPossibleValues>
                                                    <BooleanRVStates rdf:ID="false">
                                                        <hasType>
                                                            <MetaEntity rdf:ID="Boolean">
                                                                <hasType>
                                                                    <MetaEntity rdf:ID="TypeLabel">
                                                                        <hasUID rdf:datatype=
                                                                            "http://www.w3.org/2001/XMLSchema#string"

```



```

>!TypeLabel</hasUID>
<isTypeOf rdf:resource="#TypeLabel"/>
<isTypeOf rdf:resource="#Boolean"/>
<rdfs:comment rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string"
>This MetaEntity shold be assigned for the labels of all domain specific types and
subtypes.</rdfs:comment>
<isTypeOf>
<MetaEntity rdf:ID="CategoryLabel">
<hasUID rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string"
>!CategoryLabel</hasUID>
<hasType rdf:resource="#TypeLabel"/>
<rdfs:comment rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string"
>This MetaEntity should be assigned to the labels for the states of random
variables whose domain is a list of categorical values</rdfs:comment>
</MetaEntity>
</isTypeOf>
<hasType rdf:resource="#TypeLabel"/>
</MetaEntity>
</hasType>
<rdfs:comment rdf:datatype=
"http://www.w3.org/2001/XMLSchema#string"
>This MetaEntity should be applied to the truth-values T, F and  $\perp$ 
(absurd).</rdfs:comment>
<isTypeOf rdf:resource="#absurd"/>
<isTypeOf rdf:resource="#true"/>
<hasUID rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>!Boolean</hasUID>
<isTypeOf rdf:resource="#false"/>
</MetaEntity>
</hasType>
<isPossibleValueOf>
<BuiltInRV rdf:ID="forall">
<hasPossibleValues rdf:resource="#false"/>
<hasPossibleValues rdf:resource="#absurd"/>
<hasPossibleValues rdf:resource="#true"/>
</BuiltInRV>
</isPossibleValueOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>This state corresponds to the meaningfull hypotheses that have a false truth-value.
</rdfs:comment>
<isPossibleValueOf>
<BuiltInRV rdf:ID="not">
<hasPossibleValues rdf:resource="#true"/>
<hasPossibleValues rdf:resource="#absurd"/>
<hasPossibleValues rdf:resource="#false"/>
</BuiltInRV>
</isPossibleValueOf>
<isPossibleValueOf>
<BuiltInRV rdf:ID="implies">
<hasPossibleValues rdf:resource="#false"/>
<hasPossibleValues rdf:resource="#true"/>
<hasPossibleValues rdf:resource="#absurd"/>
</BuiltInRV>

```

```

</isPossibleValueOf>
<isPossibleValueOf>
  <BuiltInRV rdf:ID="iff">
    <hasPossibleValues rdf:resource="#absurd"/>
    <hasPossibleValues rdf:resource="#true"/>
    <hasPossibleValues rdf:resource="#false"/>
  </BuiltInRV>
</isPossibleValueOf>
<isPossibleValueOf rdf:resource="#or"/>
<isPossibleValueOf rdf:resource="#equalto"/>
<isPossibleValueOf>
  <BuiltInRV rdf:ID="and">
    <hasPossibleValues rdf:resource="#false"/>
    <hasPossibleValues rdf:resource="#true"/>
    <hasPossibleValues rdf:resource="#absurd"/>
  </BuiltInRV>
</isPossibleValueOf>
<isPossibleValueOf>
  <BuiltInRV rdf:ID="exists">
    <hasPossibleValues rdf:resource="#false"/>
    <hasPossibleValues rdf:resource="#true"/>
    <hasPossibleValues rdf:resource="#absurd"/>
  </BuiltInRV>
</isPossibleValueOf>
<hasUID rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>F</hasUID>
</BooleanRVStates>
</hasPossibleValues>
<hasPossibleValues rdf:resource="#absurd"/>
</BuiltInRV>
</isPossibleValueOf>
<isPossibleValueOf rdf:resource="#iff"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>This state corresponds to the meaningfull hypotheses that have a false truth-value.
</rdfs:comment>
<isPossibleValueOf rdf:resource="#and"/>
<hasType rdf:resource="#Boolean"/>
<isPossibleValueOf rdf:resource="#exists"/>
<isPossibleValueOf rdf:resource="#equalto"/>
<isPossibleValueOf rdf:resource="#not"/>
<isPossibleValueOf rdf:resource="#forall"/>
<isPossibleValueOf rdf:resource="#implies"/>
<hasUID rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>T</hasUID>
</BooleanRVStates>
</hasPossibleValues>
<hasPossibleValues rdf:resource="#false"/>
<hasPossibleValues rdf:resource="#absurd"/>
</BuiltInRV>
</isPossibleValueOf>
<isPossibleValueOf rdf:resource="#forall"/>
<hasType rdf:resource="#Boolean"/>
<isPossibleValueOf rdf:resource="#and"/>
<isPossibleValueOf rdf:resource="#implies"/>
<isPossibleValueOf rdf:resource="#iff"/>
<isPossibleValueOf rdf:resource="#or"/>

```

```

    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This state is used for the cases in which a truth-value cannot be applied to a random variable
    (i.e. meaningless, undefined, or contradictory hypotheses).</rdfs:comment>
    <hasUID rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >⊥</hasUID>
    <isPossibleValueOf rdf:resource="#exists"/>
    <isPossibleValueOf rdf:resource="#not"/>
    </BooleanRVStates>
    <BooleanRVStates rdf:about="#true"/>
    </owl:oneOf>
    </owl:Class>
    </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Resident"/>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:about="#Domain_Res"/>
  </owl:disjointWith>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasParent"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Finding_input"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasParent"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#BuiltInRV"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#Finding_MFrag"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isResidentNodeIn"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isParentOf"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >0</owl:cardinality>
    </owl:Restriction>

```

```

</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isResidentNodeIn"/>
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#Finding_MFrag"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom>
      <owl:Class>
        <owl:oneOf rdf:parseType="Collection">
          <BooleanRVStates rdf:about="#absurd"/>
          <BooleanRVStates rdf:about="#true"/>
        </owl:oneOf>
      </owl:Class>
    </owl:allValuesFrom>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >This is the subclass of class Resident (node) that includes all finding nodes. Finding nodes convey new
  evidence into a probabilistic system via a Finding_MFrag. The class is disjoint with classes Domain_res and
  BuiltInRV.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="#Node">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Entity"/>
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="isNodeFrom"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#MFrag"/>
      </owl:someValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >A node is part of an MFrag and it can be a random variable that is defined within that MFrag (a resident
    node), a RV that input values to nodes within that MFrag (an input node), or a RV that expresses the context
    in which the probability distributions within that MFrag are valid (a context node).</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>

```

```

    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="isInnerTermOf"/>
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#Node"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:someValuesFrom>
      <owl:Class rdf:about="#Entity"/>
    </owl:someValuesFrom>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#Node"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasInnerTerm"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isNodeFrom"/>
    </owl:onProperty>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#MFragment"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#ArgRelationship"/>
    </owl:allValuesFrom>
    <owl:onProperty>
      <owl:InverseFunctionalProperty rdf:about="#hasArgument"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#MetaEntity">
  <owl:disjointWith>
    <owl:Class rdf:about="#BooleanRVStates"/>
  </owl:disjointWith>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >The MetaEntity class includes all the entities that convey specific definitions about entities (e.g.
    typelabels that name the possible types of entities).</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Entity"/>
  </rdfs:subClassOf>

```

```

<owl:disjointWith rdf:resource="#ObjectEntity"/>
<owl:disjointWith>
  <owl:Class rdf:about="#CategoricalRVStates"/>
</owl:disjointWith>
<owl:equivalentClass>
  <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="isTypeOf"/>
        </owl:onProperty>
        <owl:allValuesFrom>
          <owl:Class rdf:about="#Entity"/>
        </owl:allValuesFrom>
      </owl:Restriction>
      <owl:Restriction>
        <owl:someValuesFrom>
          <owl:Class rdf:about="#Entity"/>
        </owl:someValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty rdf:about="#isTypeOf"/>
        </owl:onProperty>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:Class>
</owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="MTheory">
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#MFrag"/>
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasMFrag"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    >An MTheory is a collection of MFrams that statisfies consistency constraints ensuring the existence of a
    unique joint distribution over the random variables mentioned in the MTheory. In PR-OWL, the class
    MTheory allows a probabilistic ontology to have more than one valid MTheory to represent its RVs, and each
    individual of that class is basically a list of the MFrams that collectively form that MTheory. In addition, one
    MFram can be part of more than one MTheory.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#MFrag"/>
      </owl:someValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasMFrag"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

```

```

<owl:Class rdf:about="#ProbAssign">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasStateName"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Each cell in an PR-OWL table has a probability assignment for the state of a RV given the states of its
    parent nodes. Thus, the resulting relationship is N-ary and we opted for representing it via a reified relation
    (ProbAssign) that includes the name of the state to which the probability is being assigned, the probability
    value itself, and the list of states of parent nodes (i.e. conditionants) that collectively define the context in
    which that probability assignment is valid. Also, individuals of the ProbAssign class have an object property
    that links them with its respective PR-OWL table. </rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasStateName"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Entity"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#CondRelationship"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasConditionant"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="hasStateProb"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom rdf:resource="#CondRelationship"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasConditionant"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>

```

```

    <owl:ObjectProperty rdf:about="#hasStateName"/>
  </owl:onProperty>
  <owl:someValuesFrom>
    <owl:Class rdf:about="#Entity"/>
  </owl:someValuesFrom>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#MFrag">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasNode"/>
      </owl:onProperty>
      <owl:allValuesFrom rdf:resource="#Node"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Skolem"/>
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasSkolem"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasResidentNode"/>
      </owl:onProperty>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#Resident"/>
      </owl:allValuesFrom>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasResidentNode"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom rdf:resource="#OVariable"/>
      <owl:onProperty>
        <owl:InverseFunctionalProperty rdf:ID="hasOVariable"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</rdfs:subClassOf>

```



```

<owl:Restriction>
  <owl:onProperty>
    <owl:ObjectProperty rdf:ID="isMFragOf"/>
  </owl:onProperty>
  <owl:allValuesFrom rdf:resource="#MTheory"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:someValuesFrom>
      <owl:Class rdf:about="#Resident"/>
    </owl:someValuesFrom>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasResidentNode"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >MEBN Fragments (MFrag) are the basic structure of any MEBN logic model. MFrag represent
influences among clusters of related RVs and can portray repeated patterns using ordinary variables as
placeholders in to which entity identifiers can be substituted. In PR-OWL, each individual the MFrag class
represents a MEBN Fragment (MFrag).</rdfs:comment>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#Input"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasInputNode"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:someValuesFrom rdf:resource="#Node"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasNode"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#CategoricalRVStates">
  <owl:disjointWith rdf:resource="#MetaEntity"/>
  <owl:disjointWith rdf:resource="#ObjectEntity"/>
<rdfs:subClassOf>
  <owl:Class rdf:about="#Entity"/>
</rdfs:subClassOf>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Nodes represent random variables, which by definition have a list of mutually exclusive, collectively
exhaustive states. In PR-OWL, those states are represented by individuals from class Entity. Some random
variables have a list of categorical values as its possible states, and these are represented by elements from
subclass CategoricalRVStates.</rdfs:comment>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:hasValue rdf:resource="#CategoryLabel"/>
    <owl:onProperty>
      <owl:FunctionalProperty rdf:ID="hasType"/>
    </owl:onProperty>

```

```

    </owl:Restriction>
  </rdfs:subClassOf>
  <owl:disjointWith>
    <owl:Class rdf:about="#BooleanRVStates"/>
  </owl:disjointWith>
</owl:Class>
<owl:Class rdf:ID="SimpleArgRelationship">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Each generic random variable can have many arguments. Arguments are usually restricted in their type
    and meaning via the context nodes of an MFrag. In order to model these complex N-ary relations, PR-OWL
    makes use of the SimpleArgRelationship class, which is a reified relation that conveys the number and order
    of arguments that each RV expects, it's type (defined via a link to the TypeContext class), and the link to the
    RV itself.</rdfs:comment>
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:ID="hasArgTerm"/>
          </owl:onProperty>
          <owl:allValuesFrom rdf:resource="#OVariable"/>
        </owl:Restriction>
        <owl:Class rdf:about="#ArgRelationship"/>
        <owl:Restriction>
          <owl:onProperty>
            <owl:ObjectProperty rdf:about="#hasArgTerm"/>
          </owl:onProperty>
          <owl:someValuesFrom rdf:resource="#OVariable"/>
        </owl:Restriction>
        <owl:Restriction>
          <owl:allValuesFrom rdf:resource="#Node"/>
          <owl:onProperty>
            <owl:FunctionalProperty rdf:ID="isArgumentOf"/>
          </owl:onProperty>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:FunctionalProperty rdf:about="#isArgumentOf"/>
          </owl:onProperty>
          <owl:maxCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
            >1</owl:maxCardinality>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>
<owl:Class rdf:ID="Generative_input">
  <rdfs:subClassOf rdf:resource="#Input"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isInputInstanceOf"/>
      </owl:onProperty>
      <owl:someValuesFrom>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">

```

```

    <owl:Class rdf:about="#Domain_Res"/>
    <owl:Class rdf:about="#BuiltInRV"/>
  </owl:unionOf>
</owl:Class>
</owl:someValuesFrom>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith>
  <owl:Class rdf:about="#Finding_input"/>
</owl:disjointWith>
</owl:Class>
<owl:Class rdf:about="#Entity">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#MetaEntity"/>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#hasType"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      <owl:onProperty>
        <owl:DatatypeProperty rdf:ID="hasUID"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isConditionantOf"/>
      </owl:onProperty>
      <owl:allValuesFrom rdf:resource="#ProbAssign"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom rdf:resource="#MetaEntity"/>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#hasType"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"

```

>MEBN logic treats the world as being comprised of entities that have attributes and are related to other entities. The logic assumes uniqueness of each concept (i.e. unique name assumption), so each entity in a MEBN model has a unique identifier and no unique identifier can be assigned to more than one entity. PR-OWL follows MEBN syntax and semantics for defining entities so each member of the class Entity has a unique identifier assigned by the datatype property hasUID. OWL doesn't have the unique name assumption so the UID can be seen as a tool for providing maximum compatibility with legacy OWL ontologies, since parsers may refer to it as a means to enforce uniqueness among declared entities. It is important to note that not all concepts in an ontology have a UID, but only those which will be considered as part of the probabilistic model that is implicit in any probabilistic ontology. This structure allows mixing legacy deterministic ontologies with probabilistic ones. That allows knowledge engineers to

assign PR-OWL definitions only to the parts of the domain for which plausible reasoning is desired.</rdfs:comment>

```

<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom>
      <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
          <owl:Class rdf:about="#Node"/>
          <owl:Class rdf:about="#BuiltInRV"/>
        </owl:unionOf>
      </owl:Class>
    </owl:allValuesFrom>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="isPossibleValueOf"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:FunctionalProperty rdf:about="#hasType"/>
    </owl:onProperty>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
      >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
</owl:Class>
<owl:Class rdf:about="#Finding_input">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#Finding_MFrag"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isInputNodeIn"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isInputInstanceOf"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Finding_res"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#Finding_res"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isParentOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>

```

```

    <owl:ObjectProperty rdf:about="#isInputInstanceOf"/>
  </owl:onProperty>
  <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#Generative_input"/>
<rdfs:subClassOf rdf:resource="#Input"/>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#Finding_res"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isInputInstanceOf"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isInputNodeIn"/>
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#Finding_MFrag"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#Finding_res"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isParentOf"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isParentOf"/>
    </owl:onProperty>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:cardinality>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#DeclarativeDist">
  <owl:equivalentClass>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Restriction>
          <owl:onProperty>
            <owl:FunctionalProperty rdf:ID="hasDeclaration"/>
          </owl:onProperty>
          <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
            >1</owl:minCardinality>
        </owl:Restriction>
        <owl:Restriction>
          <owl:onProperty>
            <owl:DatatypeProperty rdf:ID="isRepresentedAs"/>

```

```

    </owl:onProperty>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:minCardinality>
  </owl:Restriction>
  <owl:Class rdf:about="#ProbDist"/>
  <owl:Restriction>
    <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >1</owl:cardinality>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isProbDistOf"/>
    </owl:onProperty>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isProbDistOf"/>
    </owl:onProperty>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#Resident"/>
    </owl:allValuesFrom>
  </owl:Restriction>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isProbDistOf"/>
    </owl:onProperty>
    <owl:someValuesFrom>
      <owl:Class rdf:about="#Resident"/>
    </owl:someValuesFrom>
  </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
</owl:equivalentClass>
<owl:disjointWith rdf:resource="#PR-OWLTable"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"

```

>A declarative distribution is a distribution that is conveyed via a xsd:string datatype, using a specific format defined in the hasDeclaration datatype property. In order to allow a 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 proprietary format.

Describing a probability distribution is a much more compact and flexible way of conveying it. However, it assumes that an OWL-P parser 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 DeclarativeDist individuals for the same RV).</rdfs:comment>

```

</owl:Class>
<owl:Class rdf:about="#Resident">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasParent"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#Resident"/>
            <owl:Class rdf:about="#Input"/>
          </owl:unionOf>
        </owl:Class>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

```

```

    </owl:Class>
    </owl:allValuesFrom>
  </owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#Input"/>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isResidentNodeIn"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="hasProbDist"/>
    </owl:onProperty>
    <owl:someValuesFrom rdf:resource="#ProbDist"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isResidentNodeIn"/>
    </owl:onProperty>
    <owl:someValuesFrom rdf:resource="#MFrag"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#MFrag"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isResidentNodeIn"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#Resident"/>
    </owl:allValuesFrom>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isParentOf"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#ProbDist"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasProbDist"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>

```

```

<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#ArgRelationship"/>
    </owl:allValuesFrom>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isArgTermIn"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:someValuesFrom rdf:resource="#Entity"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasInnerTerm"/>
    </owl:onProperty>
    <owl:allValuesFrom>
      <owl:Class rdf:about="#Resident"/>
    </owl:allValuesFrom>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="#Node"/>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:InverseFunctionalProperty rdf:about="#hasArgument"/>
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#SimpleArgRelationship"/>
  </owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#Context"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Resident nodes are the random variables that have their respective probability distribution defined in the
MFrag.</rdfs:comment>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#Entity"/>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#BooleanRVStates">
  <owl:disjointWith rdf:resource="#CategoricalRVStates"/>
  <owl:disjointWith rdf:resource="#MetaEntity"/>
  <rdfs:subClassOf rdf:resource="#Entity"/>
<rdfs:subClassOf>
  <owl:Restriction>

```



```

    <owl:hasValue rdf:resource="#Boolean"/>
    <owl:onProperty>
      <owl:FunctionalProperty rdf:about="#hasType"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<owl:disjointWith rdf:resource="#ObjectEntity"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>The BooleanRVStates class is formed by the Boolean truth-value states and are applied to Boolean
random variables.</rdfs:comment>
</owl:Class>
<owl:Class rdf:about="#Domain_Res">
  <owl:disjointWith rdf:resource="#Finding_res"/>
  <rdfs:subClassOf rdf:resource="#Resident"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>This is the subclass of class Resident (node) that includes all domain-specific resident nodes. It is
disjoint with classes Finding_res and BuiltInRV.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isResidentNodeIn"/>
      </owl:onProperty>
      <owl:someValuesFrom rdf:resource="#Domain_MFrag"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom rdf:resource="#Domain_Res"/>
    </owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isParentOf"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isResidentNodeIn"/>
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#Domain_MFrag"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:allValuesFrom rdf:resource="#Generative_input"/>
      </owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasParent"/>
      </owl:onProperty>
    </owl:Restriction>
    <owl:Class rdf:about="#Domain_Res"/>
  </owl:unionOf>
  </owl:Class>
</rdfs:subClassOf>
  <owl:disjointWith rdf:resource="#BuiltInRV"/>
</owl:Class>

```

```

<owl:Class rdf:about="#ArgRelationship">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#hasArgTerm"/>
      </owl:onProperty>
      <owl:allValuesFrom>
        <owl:Class>
          <owl:unionOf rdf:parseType="Collection">
            <owl:Class rdf:about="#OVariable"/>
            <owl:Class rdf:about="#Node"/>
            <owl:Class rdf:about="#Entity"/>
          </owl:unionOf>
        </owl:Class>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Each generic random variable can have many arguments. Arguments are usually restricted in their type
    and meaning via the context nodes of an MFrags. In order to model these complex N-ary relations, PR-OWL
    makes use of the ArgRelationship class, which is a reified relation that conveys the number and order of
    arguments that each RV expects, it's type (defined via a link to the OVariable class), and the link to the RV
    itself.
    MEBN logic has the concept of a simple and a composite random variable term. Simple RV terms accepts
    variables and constant symbols as arguments. Composite RV terms also accepts other RV terms as
    arguments. In PR-OWL, the class ArgRelationship models composite RV terms, while its
    SimpleArgRelationship subclass models simple RV terms.</rdfs:comment>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:ID="hasArgNumber"/>
      </owl:onProperty>
      <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:allValuesFrom rdf:resource="#Node"/>
      <owl:onProperty>
        <owl:FunctionalProperty rdf:about="#isArgumentOf"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Skolem">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom rdf:resource="#ArgRelationship"/>
      <owl:onProperty>
        <owl:ObjectProperty rdf:about="#isArgTermIn"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"

```

>Each individual of class Skolem represents a Skolem constant in a MEBN quantifier random variable. Each MEBN quantifier random variable corresponds to a first-order formula beginning with a universal or existential quantifier. The Skolem constant in the MEBN random variable represents a generic individual within the scope of the universal or existential quantifier of the corresponding first-order formula. MEBN logic contains a set of built-in MFragS for quantifier random variables. In PR-OWL modelers can use individuals of class Skolem to define distributions for Skolem constants used in quantifier random variables.</rdfs:comment>

```

<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#ArgRelationship"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isArgTermIn"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:ID="isSkolemIn"/>
    </owl:onProperty>
    <owl:allValuesFrom rdf:resource="#MFrag"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:allValuesFrom rdf:resource="#OVariable"/>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#representsOVar"/>
    </owl:onProperty>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#representsOVar"/>
    </owl:onProperty>
    <owl:someValuesFrom rdf:resource="#OVariable"/>
  </owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty>
      <owl:ObjectProperty rdf:about="#isSkolemIn"/>
    </owl:onProperty>
    <owl:someValuesFrom rdf:resource="#MFrag"/>
  </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
<owl:ObjectProperty rdf:about="#hasContextNode">
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#hasNode"/>
  </rdfs:subPropertyOf>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isContextNodeIn"/>
  </owl:inverseOf>

```

```

<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>This object property links an MFRag to the context nodes being applied to it.
The inverse property is isContextIn</rdfs:comment>
<rdfs:range rdf:resource="#Context"/>
<rdfs:domain rdf:resource="#Domain_MFrag"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasInnerTerm">
<rdfs:range rdf:resource="#Node"/>
<owl:inverseOf>
<owl:ObjectProperty rdf:about="#isInnerTermOf"/>
</owl:inverseOf>
<rdfs:domain rdf:resource="#Node"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>This object property makes the connection between the many possible inner terms inside a MENB
equation. It is used to decompose random variable terms usually employed in context and input nodes.
The inverse property is isInnerTermOf.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isInputNodeIn">
<rdfs:subPropertyOf>
<owl:ObjectProperty rdf:about="#isNodeFrom"/>
</rdfs:subPropertyOf>
<rdfs:domain rdf:resource="#Input"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>This object property links a node to the MFrag that have it as an input.
The inverse property is hasInputNode.</rdfs:comment>
<owl:inverseOf>
<owl:ObjectProperty rdf:about="#hasInputNode"/>
</owl:inverseOf>
<rdfs:range rdf:resource="#MFrag"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isParentOf">
<owl:inverseOf>
<owl:ObjectProperty rdf:about="#hasParent"/>
</owl:inverseOf>
<rdfs:range rdf:resource="#Resident"/>
<rdfs:domain>
<owl:Class>
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Resident"/>
<owl:Class rdf:about="#Input"/>
</owl:unionOf>
</owl:Class>
</rdfs:domain>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>This object property links a resident or input node of an MFrag with its respective children, which are
resident nodes in that same MFrag.
The inverse property is hasParent.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isMFragOf">
<owl:inverseOf>
<owl:ObjectProperty rdf:about="#hasMFrag"/>
</owl:inverseOf>
<rdfs:domain rdf:resource="#MFrag"/>
<rdfs:range rdf:resource="#MTheory"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"

```

>This object property links one MFrag to one or more MTheories (i.e. individuals of class MTheory) that have that MFrag as its component.

The inverse property is hasMFrag.</rdfs:comment>

```
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasInputNode">
  <rdfs:range rdf:resource="#Input"/>
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#hasNode"/>
  </rdfs:subPropertyOf>
  <owl:inverseOf rdf:resource="#isInputNodeIn"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

>This object property links each MFrag with its respective input nodes.

The inverse property is isInputNodeIn.</rdfs:comment>

```
<rdfs:domain rdf:resource="#MFrag"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasArgTerm">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

>This object property links one instance of class ArgRelationship (which is linked to a RV) to an internal variable within the home MFrag where its RV is resident, to a node that is being used as argument in that RV, or to a MEBN entity.

This object property is the inverse of isArgTermIn.

One individual of the class ArgRelationship can have only one RV (since it refers to a specific argument of an RV), and thus can be related to only one OVariable (Simple RV Terms) or one Node (Composite RV Terms), which makes that property a functional one.

The inverse property is isArgTermIn.</rdfs:comment>

```
<rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
<owl:inverseOf>
  <owl:ObjectProperty rdf:about="#isArgTermIn"/>
</owl:inverseOf>
<rdfs:range>
  <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Entity"/>
      <owl:Class rdf:about="#OVariable"/>
      <owl:Class rdf:about="#Resident"/>
      <owl:Class rdf:about="#Skolem"/>
    </owl:unionOf>
  </owl:Class>
</rdfs:range>
<rdfs:domain rdf:resource="#ArgRelationship"/>
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:ID="isProbAssignIn">
  <rdfs:domain rdf:resource="#ProbAssign"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

>This is the inverse of the hasProbAssign object property and links one individual probability assignment to its respective probability distribution table.</rdfs:comment>

```
<rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
<rdfs:range rdf:resource="#PR-OWLTable"/>
<owl:inverseOf>
  <owl:InverseFunctionalProperty rdf:about="#hasProbAssign"/>
</owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasInputInstance">
  <rdfs:range rdf:resource="#Input"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isInputInstanceOf"/>
```

```

</owl:inverseOf>
<rdfs:domain>
  <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Resident"/>
      <owl:Class rdf:about="#BuiltInRV"/>
    </owl:unionOf>
  </owl:Class>
</rdfs:domain>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >This object property links a resident node or a built-in RV to its many possible "input node instances", or
  the instances of input nodes that take their values from that resident node or built-in RV.
  The inverse property is isInputInstanceOf. </rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isContextNodeIn">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links one context node to the respective MFragment in which that context node applies.
    The inverse property is hasContext.</rdfs:comment>
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#isNodeFrom"/>
  </rdfs:subPropertyOf>
  <rdfs:domain rdf:resource="#Context"/>
  <rdfs:range rdf:resource="#Domain_MFragment"/>
  <owl:inverseOf rdf:resource="#hasContextNode"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasProbDist">
  <rdfs:range rdf:resource="#ProbDist"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isProbDistOf"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Resident"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links a RV to its respective probability distributions, as defined in that RV's home
    MFrags. Note that this property is not being defined as functional, implying a polymorphic version of MEBN
    (where each RV can have different distributions in different MFrags).
    The inverse of this property is isProbDistOf.</rdfs:comment>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#isSkolemIn">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >This object property relates one Skolem constant (i.e. an individual from class Skolem) to the MFragment in
      which it is defined.
      The inverse of this property is hasSkolem.</rdfs:comment>
    <rdfs:range rdf:resource="#MFragment"/>
    <owl:inverseOf>
      <owl:ObjectProperty rdf:about="#hasSkolem"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="#Skolem"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="#isConditionantOf">
    <rdfs:domain rdf:resource="#CondRelationship"/>
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >This object property links one possible state of a parent node to the configuration that is conditioning its
      children state's probability distribution.
      The inverse property is hasConditionant.</rdfs:comment>
    <owl:inverseOf>

```

```

    <owl:ObjectProperty rdf:about="#hasConditionant"/>
  </owl:inverseOf>
  <rdfs:range rdf:resource="#ProbAssign"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isPossibleValueOf">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasPossibleValues"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Entity"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Node"/>
        <owl:Class rdf:about="#BuiltInRV"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property correlates one entity with the node(s) of one or more MFrag(s) that have such entity
    as a possible state.

```

Note that the individuals listed as being possible values of a node must form a mutually exclusive, collectively exhaustive set. PR-OWL has the same tools for enforcing exclusiveness (i.e. MEBN entity unique name assumption and the existential and universal qualifiers acting together as a closure axiom), but the domain expert must ensure completeness.

The inverse property is hasPossibleValues.</rdfs:comment>

```

</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isResidentNodeIn">
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#hasResidentNode"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Resident"/>
  <rdfs:range rdf:resource="#MFrag"/>
  <rdfs:subPropertyOf>
    <owl:ObjectProperty rdf:about="#isNodeFrom"/>
  </rdfs:subPropertyOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links an individual of class Node to the MFrag(s) that have this node as a resident
    node.

```

The inverse property is hasResidentNode</rdfs:comment>

```

</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasParent">
  <rdfs:domain rdf:resource="#Resident"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Resident"/>
        <owl:Class rdf:about="#Input"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links a resident node of an MFrag with its respective parent(s), which has(have) to
    be an individual of either the class Resident or the class Input.

```

The inverse property is isParentOf.

```

</rdfs:comment>
  <owl:inverseOf rdf:resource="#isParentOf"/>

```

```

</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#representsOVar">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links a Skolem constant (i.e. an individual of class Skolem) to the ordinary variable it
    represents in a quantifier expression. The property is functional since each Skolem constant represents only
    one ordinary variable in the model.
  </rdfs:comment>
  The inverse property is isRepBySkolem.</rdfs:comment>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="isRepBySkolem"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#Skolem"/>
  <rdfs:range rdf:resource="#OVariable"/>
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasSkolem">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property relates one MFragment with the Skolem constants (i.e. an individual from class Skolem)
    that are defined in that MFragment.
  </rdfs:comment>
  The inverse of this property is isSkolemIn.</rdfs:comment>
  <rdfs:domain rdf:resource="#MFragment"/>
  <owl:inverseOf rdf:resource="#isSkolemIn"/>
  <rdfs:range rdf:resource="#Skolem"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasNode">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links one MFragment with its nodes.
  </rdfs:comment>
  The inverse property is isNodeFrom.</rdfs:comment>
  <rdfs:domain rdf:resource="#MFragment"/>
  <rdfs:range rdf:resource="#Node"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:about="#isNodeFrom"/>
  </owl:inverseOf>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isRepBySkolem">
  <rdfs:range rdf:resource="#Skolem"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links one ordinary variable to the Skolem constant that represents that ordinary
    variable in quantified expressions. The property is inverse functional, since one Skolem constant can
    represent only the group of entities that can be replaced with that ordinary variable in the model.
  </rdfs:comment>
  The inverse property is representsOVar.</rdfs:comment>
  <rdfs:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  <owl:inverseOf rdf:resource="#representsOVar"/>
  <rdfs:domain rdf:resource="#OVariable"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasPossibleValues">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property defines what are the possible values of a node in an MFragment (which is by definition a
    random variable). Possible states include all kinds of entities.
  </rdfs:comment>
  The inverse property is isPossibleValueOf.</rdfs:comment>
  <owl:inverseOf rdf:resource="#isPossibleValueOf"/>
  <rdfs:range rdf:resource="#Entity"/>
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#BuiltInRV"/>
        <owl:Class rdf:about="#Node"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>

```



```

    </owl:unionOf>
  </owl:Class>
</rdfs:domain>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isNodeFrom">
  <rdfs:range rdf:resource="#MFragment"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This general object property links one node to the MFragment it belongs to.
  </rdfs:comment>
  The inverse property is hasNode</rdfs:comment>
  <rdfs:domain rdf:resource="#Node"/>
  <owl:inverseOf rdf:resource="#hasNode"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isInputInstanceOf">
  <rdfs:domain rdf:resource="#Input"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links an input node to its "generative resident node", or the resident node to which
  that input node is a copy.
  The inverse property is hasInputInstance.</rdfs:comment>
  <owl:inverseOf rdf:resource="#hasInputInstance"/>
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Resident"/>
        <owl:Class rdf:about="#BuiltInRV"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasResidentNode">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links an MFragment with its respective resident node(s).
  The inverse property is isResidentNodeIn</rdfs:comment>
  <rdfs:subPropertyOf rdf:resource="#hasNode"/>
  <rdfs:range rdf:resource="#Resident"/>
  <owl:inverseOf rdf:resource="#isResidentNodeIn"/>
  <rdfs:domain rdf:resource="#MFragment"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isContextInstanceOf">
  <rdfs:range>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Domain_Res"/>
        <owl:Class rdf:about="#BuiltInRV"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:range>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links context node to its "generative resident node" or built-in RV (i.e. the resident
  node or built-in RV from which the context node is a pointer).
  The inverse property is hasContextInstance</rdfs:comment>
  <rdfs:domain rdf:resource="#Context"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <owl:inverseOf>
    <owl:ObjectProperty rdf:ID="hasContextInstance"/>
  </owl:inverseOf>

```

```

<owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasMFrag">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >This object property links one MTheory with its respective MFrag. Usually, a probabilistic ontology will
have only one MTheory as a means to convey the global joint probability distribution of its random variables.
However, MEBN logic allows many possible MTheories to represent a given domain, so it is reasonable to
infer that in some circumstances it might be preferable to have one probability ontology being represented by
more than one MTheory.
The inverse property is isMFragOf</rdfs:comment>
  <rdfs:domain rdf:resource="#MTheory"/>
  <owl:inverseOf rdf:resource="#isMFragOf"/>
  <rdfs:range rdf:resource="#MFrag"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasContextInstance">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Domain_Res"/>
        <owl:Class rdf:about="#BuiltInRV"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >This object property links a resident node or a built-in RV to its many possible "context node instances",
or the instances of context nodes that take their values from that resident node or built-in RV.
The inverse property is isContextInstanceOf. </rdfs:comment>
  <rdfs:range rdf:resource="#Context"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
  <owl:inverseOf rdf:resource="#isContextInstanceOf"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isOVariableIn">
  <owl:inverseOf>
    <owl:InverseFunctionalProperty rdf:about="#hasOVariable"/>
  </owl:inverseOf>
  <rdfs:domain rdf:resource="#OVariable"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
  <rdfs:range rdf:resource="#MFrag"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >This functional object property relates one ordinary variable (i.e. an individual from class OVariable) to its
respective MFrag.
The inverse of this property is hasOVariable.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#isProbDistOf">
  <rdfs:domain rdf:resource="#ProbDist"/>
  <rdfs:range rdf:resource="#Resident"/>
  <owl:inverseOf rdf:resource="#hasProbDist"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >This object property links a probability distribution to its respective RV (resident node). Note that this
property is functional, since each probability distribution in a MFrag defines a unique RV.
The inverse of this property is hasProbDist.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasConditionant">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Each instance of the class ProbAssign corresponds to the probability assignment for a given state of a
RV. This probability assignment is conditioned by the parent RVs of that RV. This object property conveys

```

the list of the states of the parent RV which have influenced that specific probability assignment. Since any MEBN entity can be a state in a RV, this property has MEBNEntity class as its range.

The inverse property is isConditionantOf</rdfs:comment>

```
<owl:inverseOf rdf:resource="#isConditionantOf"/>
```

```
<rdfs:range rdf:resource="#CondRelationship"/>
```

```
<rdfs:domain rdf:resource="#ProbAssign"/>
```

```
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:about="#isTypeOf">
```

```
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>This is the inverse of hasType object property, and basically lists all the MEBN entities that have its respective type defined by that specific individual of either the MetaEntity class or the ObjectEntity class.</rdfs:comment>

```
<rdfs:domain rdf:resource="#MetaEntity"/>
```

```
<rdfs:range rdf:resource="#Entity"/>
```

```
<owl:inverseOf>
```

```
<owl:FunctionalProperty rdf:about="#hasType"/>
```

```
</owl:inverseOf>
```

```
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
```

```
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:about="#hasParentName">
```

```
<rdfs:range rdf:resource="#Node"/>
```

```
<rdfs:domain rdf:resource="#CondRelationship"/>
```

```
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>This object property links a CondRelationship to a Node. The reified conditional relationship is used to build PR-OWL Tables. One table usually has many probability assignments (which correspond to cells in a table), and each probability assignment has a set of conditionants. Conditionants are the states of the parents of a node that form a combination where a given probability assignment holds. Each CondRelationship defines a pair parent/state-of-parent, and the hasParentName property defines the parent name of that pair. </rdfs:comment>

```
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
```

```
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:about="#isArgTermIn">
```

```
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
```

```
<owl:inverseOf rdf:resource="#hasArgTerm"/>
```

```
<rdfs:domain>
```

```
<owl:Class>
```

```
<owl:unionOf rdf:parseType="Collection">
```

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

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

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

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

```
</owl:unionOf>
```

```
</owl:Class>
```

```
</rdfs:domain>
```

```
<rdfs:range rdf:resource="#ArgRelationship"/>
```

```
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>This object property links an individual of class OVariable, Resident, Entity, or Skolem to one ArgRelationship(s) that has individual as its argument. Each ArgRelationship can have only one argument, but each individual of those classes can refer to many ArgRelationships.

The inverse of this property is hasArgTerm.</rdfs:comment>

```
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:about="#isInnerTermOf">
```

```
<owl:inverseOf rdf:resource="#hasInnerTerm"/>
```

```
<rdfs:range rdf:resource="#Node"/>
```

```
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>This object property is used to model expressions.

The inverse property is hasInnerTerm</rdfs:comment>

```
<rdfs:domain rdf:resource="#Node"/>
```

```
</owl:ObjectProperty>
```

```
<owl:ObjectProperty rdf:about="#hasStateName">
```

```
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>When a probability distribution is conveyed as an PR-OWL table, each individual cell is represented as an individual of the ProbAssign class. This object property refers to which state of a random variable (i.e. MFrag node) a given probability assignment refers to.

The property itself is functional, since one state can have only one probability assignment for the configuration listed in each individual of the ProbAssign class.</rdfs:comment>

```
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
```

```
<rdfs:range rdf:resource="#Entity"/>
```

```
<rdfs:domain rdf:resource="#ProbAssign"/>
```

```
</owl:ObjectProperty>
```

```
<owl:DatatypeProperty rdf:about="#hasUID">
```

```
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>MEBN logic has the unique naming assumption, which is not assumed in OWL (even though tools such as Protege make that assumption for improved reasoning purposes). In order to make sure that a tool that does not assume unique identifies would not prevent MEBN reasoners to work, each MEBN entity has a unique identifier assigned by this datatype property.

The UID itself is conveyed as a xsd:string, and the hasUID datatype property is declared as functional in order to enforce uniqueness.</rdfs:comment>

```
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
```

```
<rdfs:domain rdf:resource="#Entity"/>
```

```
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
```

```
</owl:DatatypeProperty>
```

```
<owl:DatatypeProperty rdf:about="#isRepresentedAs">
```

```
<rdfs:domain rdf:resource="#DeclarativeDist"/>
```

```
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>This datatype property defines how a given declarative probability distribution is expressed. Each probability distribution can be expressed in different formats, and each format is defined by this datatype property. Possible formats include Netica tables, Netica equations, Quiddity formulas, MEBN syntax, and others. However, the declaration itself is stored in the hasDeclaration datatype property as a string so parsers will have to know how to deal with the specific text format of each declaration.</rdfs:comment>

```
<rdfs:range>
```

```
<owl:DataRange>
```

```
<owl:oneOf rdf:parseType="Resource">
```

```
<rdf:rest rdf:parseType="Resource">
```

```
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
>MS_DSC</rdf:first>
```

```
<rdf:rest rdf:parseType="Resource">
```

```
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
>Quiddity_SPI</rdf:first>
```

```
<rdf:rest rdf:parseType="Resource">
```

```
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
>Netica_DNE</rdf:first>
```

```
<rdf:rest rdf:parseType="Resource">
```

```
<rdf:rest rdf:parseType="Resource">
```

```
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
>BIF</rdf:first>
```

```
<rdf:rest rdf:parseType="Resource">
```

```
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
>Hugin_NET</rdf:first>
```

```
<rdf:rest rdf:parseType="Resource">
```

```
<rdf:rest rdf:parseType="Resource">
```

```
<rdf:rest rdf:parseType="Resource">
```

```

<rdf:rest rdf:parseType="Resource">
  <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >Other</rdf:first>
  <rdf:rest rdf:parseType="Resource">
    <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >PR-OWL_MEBN</rdf:first>
    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
  </rdf:rest>
</rdf:rest>
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Ergo_ENT</rdf:first>
</rdf:rest>
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >IDEAL_IDE</rdf:first>
</rdf:rest>
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >Genie_DSL</rdf:first>
</rdf:rest>
</rdf:rest>
</rdf:rest>
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >MS_XBN</rdf:first>
</rdf:rest>
</rdf:rest>
</rdf:rest>
<rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >BNIF</rdf:rest>
</owl:oneOf>
</owl:DataRange>
</rdfs:range>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
</owl:DatatypeProperty>
<owl:FunctionalProperty rdf:about="#isSubsBy">
  <rdfs:range rdf:resource="#MetaEntity"/>
  <rdfs:domain rdf:resource="#OVariable"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links one instance of class OVariable to type of the entity that can substitute it. Each
    argument of a RV has its expected type defined within the home MFrag of that RV. In PR-OWL, the type
    restrictions are defined directly through the OVariable using the isSubsBy property. One MFrag can have
    many OVariables (which can be themselves linked to many SimpleArgRelationships) but each OVariable
    has a unique type, which is explicitly defined by the type of the entity that can substitute that OVariable.
    This object property is the inverse of subsOVar.</rdfs:comment>
  <owl:inverseOf>
    <owl:InverseFunctionalProperty rdf:about="#subsOVar"/>
  </owl:inverseOf>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasParentState">
  <rdfs:domain rdf:resource="#CondRelationship"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <rdfs:range rdf:resource="#Entity"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >This object property links a CondRelationship to an Entity. The reified conditional relationship is used to
    build PR-OWL Tables. One table usually has many probability assignments (which correspond to cells in a
    table), and each probability assignment has a set of conditionants. Conditionants are the states of the

```

parents of a node that form a combination where a given probability assignment holds. Each CondRelationship defines a pair parent/state-of-parent, and the hasParentState property defines the parent state of that pair.</rdfs:comment>

```
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#isArgumentOf">
  <rdfs:range rdf:resource="#Node"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
  <owl:inverseOf>
    <owl:InverseFunctionalProperty rdf:about="#hasArgument"/>
  </owl:inverseOf>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

>This object property links an Argument Relationship to its respective Node (i.e. to the individual of class Node that has this ArgRelationship into its argument list).

The inverse of this property is hasArgument.

```
</rdfs:comment>
```

```
<rdfs:domain rdf:resource="#ArgRelationship"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasType">
  <owl:inverseOf rdf:resource="#isTypeOf"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

>In the extended MEBN logic that is the backbone of PR-OWL, each and every entity has a type. The list of types consists of the individuals from class MetaEntity.

This functional object property defines the type of each entity by linking it to an individual of the MetaEntity class.

Every entity has a MetaEntity (TypeLabel, CategoryLabel, Boolean, or a domain-specific label) as a Type. As an example, an hypothetical individual of an ObjectEntity class named Starship would have type Starship, which is a domain-specific label for an ObjectEntity individual that happens to be a starship. That domain-specific label is itself an individual of the MetaEntity class.

The inverse property is isTypeOf.</rdfs:comment>

```
<rdfs:domain rdf:resource="#Entity"/>
<rdfs:range rdf:resource="#MetaEntity"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasDeclaration">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

>This datatype property conveys the declarative probability distributions. Each probability distribution can be expressed in different formats and each format is defined by the datatype property isRepresentedAs.

Possible formats include Netica tables, Netica equations, Quiddity formulas, MEBN syntax, and others.

However, the declaration itself is stored as a string so parsers are expected to understand how to deal with the specific text format of each declaration.</rdfs:comment>

```
<rdfs:domain rdf:resource="#DeclarativeDist"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasStateProb">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

>This datatype property is used to store the actual probability of an individual ProbAssign. Currently, OWL has no support for user defined datatypes, so instead of using owl-p:prob datatype (which includes all decimals between 0 and 1 inclusive) we are using xsd:decimal for compatibility purposes.</rdfs:comment>

```
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
<rdfs:domain rdf:resource="#ProbAssign"/>
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#decimal"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:ID="isDefault">
  <rdfs:domain rdf:resource="#ProbDist"/>
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
```

>This datatype property indicates whether a probability distribution is the default probability distribution of a node or not. Default probability distributions for nodes are used when the context nodes of the MFRag containing those nodes are not met.</rdfs:comment>

```
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#boolean"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:FunctionalProperty rdf:about="#hasArgNumber">
<rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
<rdfs:domain rdf:resource="#ArgRelationship"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>This datatype property assigns the argument number of an argument relationship. As an example, if we have a random variable with 3 arguments, it will have three ArgRelationship reified relations. The first argument of the RV will have the number 1 assigned to its respective hasArgNumber property, the second will have the number 2 assigned and the third will have the number 3 assigned. In short this property keeps track of the ordering between the arguments of an RV.

The datatype itself is a nonNegativeInteger. We used this instead of a positiveInteger because we wanted zero as a possible value, since we assume that a RV with no arguments means a global RV.</rdfs:comment>

```
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
</owl:FunctionalProperty>
<owl:InverseFunctionalProperty rdf:about="#hasArgument">
<rdfs:range rdf:resource="#ArgRelationship"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>This object property is the link between a node in an MFRag and the reified relation that conveys its respective arguments. Note that each instance of a node will have only one argument relationship, which is defined within that node's MFRag.

The inverse of this property is isArgumentOf</rdfs:comment>

```
<owl:inverseOf rdf:resource="#isArgumentOf"/>
<rdfs:domain rdf:resource="#Node"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:InverseFunctionalProperty>
<owl:InverseFunctionalProperty rdf:about="#hasProbAssign">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>An PR-OWL table is formed by many individual members of the class ProbAssign, which are cells in that table. This object property relates one PR-OWL table to its respective cells (ProbAssign elements).

The inverse property is isProbAssignIn.</rdfs:comment>

```
<rdfs:range rdf:resource="#ProbAssign"/>
<owl:inverseOf rdf:resource="#isProbAssignIn"/>
<rdfs:domain rdf:resource="#PR-OWLTable"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:InverseFunctionalProperty>
<owl:InverseFunctionalProperty rdf:about="#subsOVar">
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>This object property assigns MetaEntity individuals in order to define the type of the substituters for each MFRag ordinary variable.

Its inverse property is the functional isSubsBy.</rdfs:comment>

```
<owl:inverseOf rdf:resource="#isSubsBy"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
<rdfs:domain rdf:resource="#MetaEntity"/>
<rdfs:range rdf:resource="#OVariable"/>
</owl:InverseFunctionalProperty>
<owl:InverseFunctionalProperty rdf:about="#hasOVariable">
<rdfs:domain rdf:resource="#MFRag"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

>This inverse functional object property relates one MFRag to its ordinary variables (i.e. individuals from class OVariable that are related to this MFRag).

```

The inverse of this property is isOVariableIn.</rdfs:comment>
  <owl:inverseOf rdf:resource="#isOVariableIn"/>
  <rdfs:range rdf:resource="#OVariable"/>
  <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty"/>
</owl:InverseFunctionalProperty>
<owl:DataRange>
  <owl:oneOf rdf:parseType="Resource">
    <rdf:rest rdf:parseType="Resource">
      <rdf:rest rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:parseType="Resource">
            <rdf:rest rdf:parseType="Resource">
              <rdf:rest rdf:parseType="Resource">
                <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                >indref</rdf:first>
                <rdf:rest rdf:parseType="Resource">
                  <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                  >implies</rdf:first>
                  <rdf:rest rdf:parseType="Resource">
                    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                    <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                    >equalto</rdf:first>
                  </rdf:rest>
                </rdf:rest>
              </rdf:rest>
            </rdf:rest>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >exists</rdf:first>
        </rdf:rest>
      <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >iff</rdf:first>
    </rdf:rest>
  <rdf:rest rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >forall</rdf:rest>
</rdf:rest>
  <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >not</rdf:rest>
</rdf:rest>
  <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >or</rdf:rest>
</rdf:rest>
  <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >and</rdf:rest>
</owl:oneOf>
</owl:DataRange>
<rdf:Description>
  <owl:allValuesFrom rdf:resource="#Domain_MFrag"/>
  <owl:onProperty rdf:resource="#isInputNodeIn"/>
</rdf:Description>
</rdf:RDF>

```



## Appendix C Potential Applications for PR-OWL Outside the Semantic Web

This appendix explores two important application scenarios for probabilistic ontologies among the ones in which PR-OWL has a great potential to be employed outside the scope of the Semantic Web.

The first one, the DTB project, covers the ontology mapping problem, where the ontologies work as information brokers between each distinct software application being used in the system and a domain-free, probabilistic meta-ontology, dubbed IO (Integration Ontology). IO carries only information about the semantic mapping between the concepts of each ontology. The information represented by IO consists of probabilistic statements about the relationship between terms in the domain ontologies.

The second example, the Wise Pilot system, explores the difficulties of performing multi-sensor data fusion with common BNs, the feasibility of using MEBN logic in that problem, and proposes the use of PR-OWL as a means to achieve sensor interoperability and information sharing between combatant platforms in a tactical environment.

### C.1 PR-OWL for Integration Ontologies: The DTB Project

DTB stands for Detection of Threat Behavior, an ongoing project initially funded by ARDA<sup>30</sup> and conducted by IET<sup>31</sup> and GMU. The project focused on a particularly insidious threat: that posed by individuals who misuse their privileges to gain access to sensitive information in order to make it available to unauthorized parties (e.g.: other states, terrorists), or to manipulate it with the purpose of producing misleading analysis (Alghamdi *et al.*, 2004).

The overall idea of the DTB project is to model user queries and detect situations in which users in sensitive positions may be accessing documents outside their assigned areas of responsibility. This novel approach to insider threats assumes a controlled environment in which rules for accessing information are clearly defined and, ideally, tightly enforced.

Although such environments provide little encouragement to insider threats, unusual access patterns are not easily detected given current technology. In fact,

---

<sup>30</sup> ARDA – Advanced Research and Development Activity ([www.ic-arda.org](http://www.ic-arda.org))

<sup>31</sup> IET – Information Extraction and Transport, Inc. ([www.iet.com](http://www.iet.com))

documented cases in which insiders using unsophisticated tactics to outsmart standard security systems (e.g., CNN.com, 1998, 2001) leave a very uncomfortable open question: how about the sophisticated ones?

Catching more elaborate patterns that might be characteristic of users attempting illegal activities such as disclosure of classified information is a daunting task that must be tackled with a powerful inference method capable of dealing with the uncertainty involved in the process.

The flexible modeling framework provided by multi-entity Bayesian networks make it an obvious candidate to model the intricacies of security-controlled environments. Its natural ability to capture a domain with the richness of details required for feeding its inference engine is a major strength, but poses a well know challenge to modelers: how to make the model interoperable among different agencies.

This requirement implies conflicting objectives. Initially, there's a quest for being precise enough to capture the subtlest hints of wrong behavior under a given agency's rules. Yet, there is also the need for constructing a model that is general enough to be suitable to other agencies. This is a trade-off nightmare to most modeling techniques, and an issue that was also perceived in the DTB project.

The project's final product is supposed to deal with a community with many possible users, both inside the Intel community and outside it. This leads to diverse (although similar) vocabularies, policies, organization culture, etc, with a great potential of rendering the model assumptions imprecise at best. Like almost all complex domains the Intelligence community does not have a commonly accepted conceptualization of its rules, policies, or vocabulary; deeming sophisticated, detail-rich systems unlikely to achieve interoperability without extra effort devoted specifically to this end.

In the DTB project, extra effort was devoted to the heavy use of ontologies. Because different views of a domain have to be represented by different ontologies, any interoperable system built upon ontologies must have a means of dealing with the ontology mapping problem.

The project is in its initial part, where the focus is on building the behavioral model and on devising data mining algorithms capable of extracting the document relevance data that will feed that model. At the same time, two ontologies were made as a way of capturing the subtleties of both the MEBN model and the data mining algorithms.

Both ontologies were developed with the open source software Protégé. The first, the *Insider behavior ontology (IB)*, describes the MEBN model of insider threat behavior. Figure 36 depicts the IB ontology. The second ontology, the *Organization and Task Ontology (OT)*, is shown in Figure 37 and portrays the various aspects of an internal organization. Among those we can cite its internal rules, details such as "need to know", individual clearance, and compartment type (and these terms' respective meanings with regard to data access), the data mining algorithms we use to capture document relevance, and other particularities of the Intelligence domain.

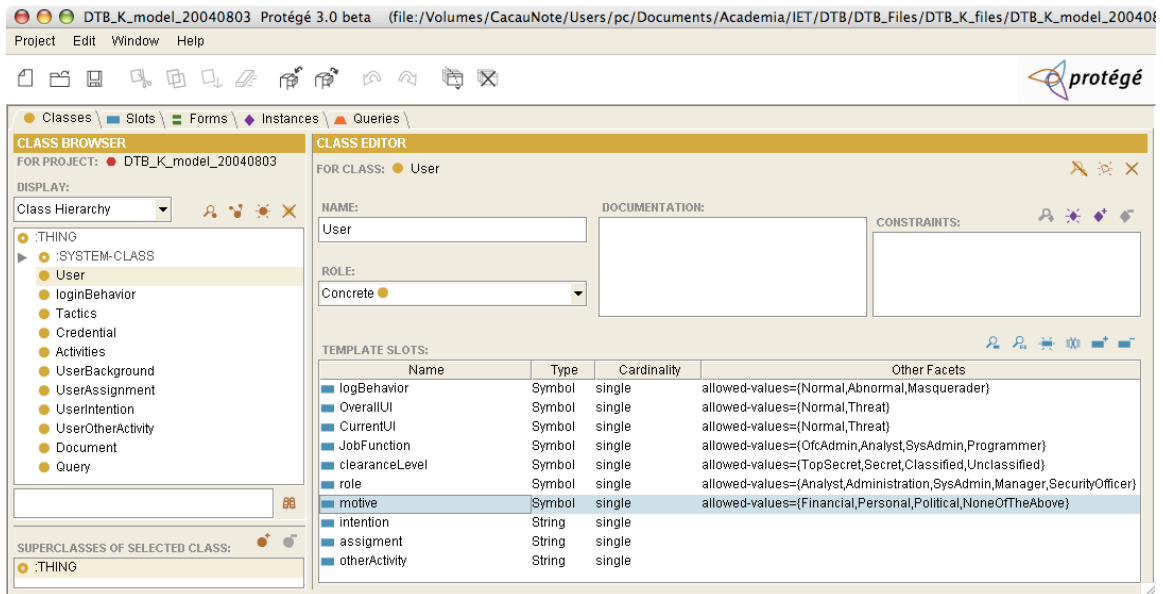


Figure 36. The Insider Behavior Ontology (IB)

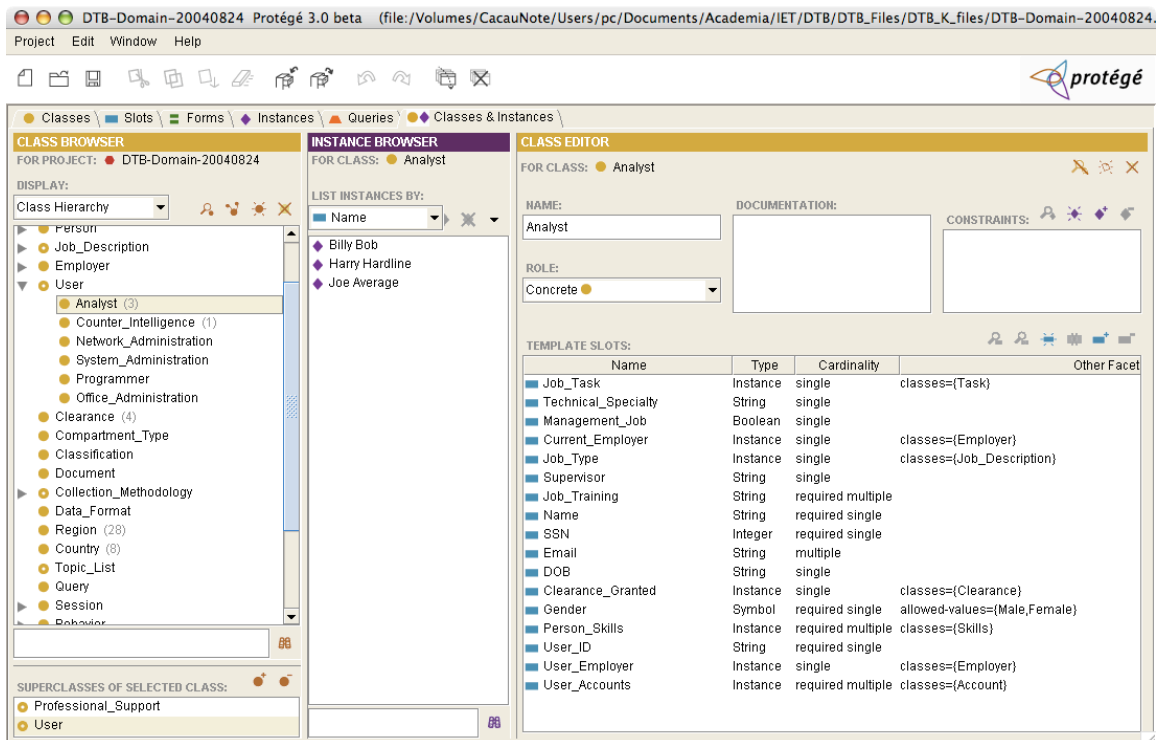


Figure 37. The Organization and Task Ontology (OT)

It is important to emphasize the role of ontologies as a tool for enforcing the semantic consistency of the models. Since both ontologies were made “in sync” with the development phase of their subjects, the modelers were forced not only to think about the

specific model details but also about ensuring that each variable and its semantic meaning is consistent with the models and with the domain's concepts.

Once the behavioral model and the data mining algorithms were ready, the next step was to extract the actual data to be used for assessing the relevance of each user's search with respect to his/her assigned task. For this task, the project adopted the software Glass Box, a Java-based user monitoring application available to researchers on ARDA's Novel Intelligence from Massive Data project (NIMD) at <http://glassbox.labworks.org>. Glass Box was used to capture the actions of users and then extract the information needed for the data mining algorithms. The overall process is depicted in Figure 38.



Figure 38. The Insider Threat Detection Process – Initial Setup

The flow starts with a set of users (top left), from whom their queries and general system usage is analyzed. User queries are stored for being processed by the data mining algorithms that will extract selected parameters regarding search relevance, which will feed the behavioral MEBN model. System usage refers to general parameters that can be used by the MEBN model to make inferences about unusual patterns (e.g. user login time, copy and paste, etc).

Figure 39 shows the same setup viewed from the perspective of the software modules being used and respective integration requirements. Here we see that Glass Box is used for capturing all data, where some will be discharged, some will feed the MEBN model directly, and some will go through the data mining process, which will capture the relevance of a given user's searches with respect to his/her assigned task. The results of the data mining process will also feed the behavioral model, as further information for assessing each user's likelihood of being an insider threat or not.

As highlighted above, the communication between each software package has to be "hardwired" via their respective APIs, in a tedious, manual, expensive, error-prone process that has to be repeated for every change in any parameter of any software package, for any additional feature in the system, and for every change of policy inside the agency in which the system is installed.

This inflexible scheme also hinders interoperability; since for each and every agency where the system is to be installed we would have to go through the whole process again and little if any of the previous setup efforts can be used in a new one.

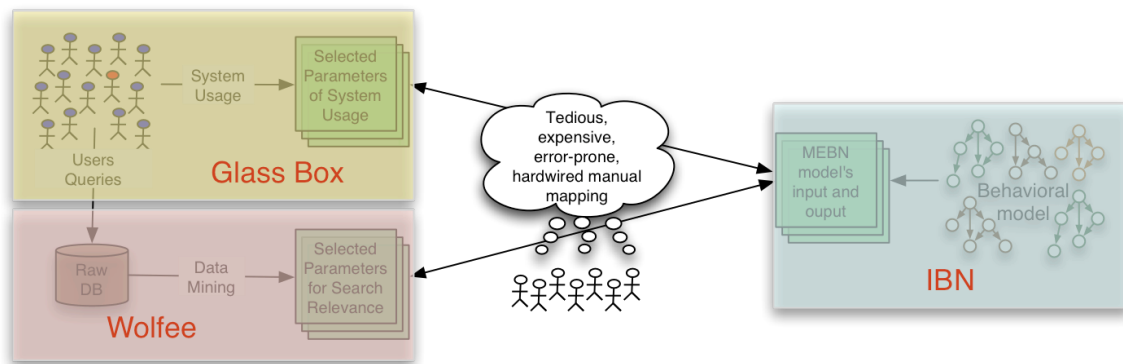


Figure 39. The Insider Threat Detection Process – Data Interchange

The approach used for solving the DTB project's interoperability problem was to use ontologies as information brokers between each distinct software application in the system. By doing so, the parameters are “hardwired” between each application and its respective ontology, instead of between applications as in the original scheme depicted in Figure 39.

Therefore, if (say) there were a change in working hours of a given agency, there would be no need to search all APIs for variables using this information. Instead, only a unique ontology has to be changed and the others will be updated via the Integration Ontology (IO). This process is shown in Figure 40.

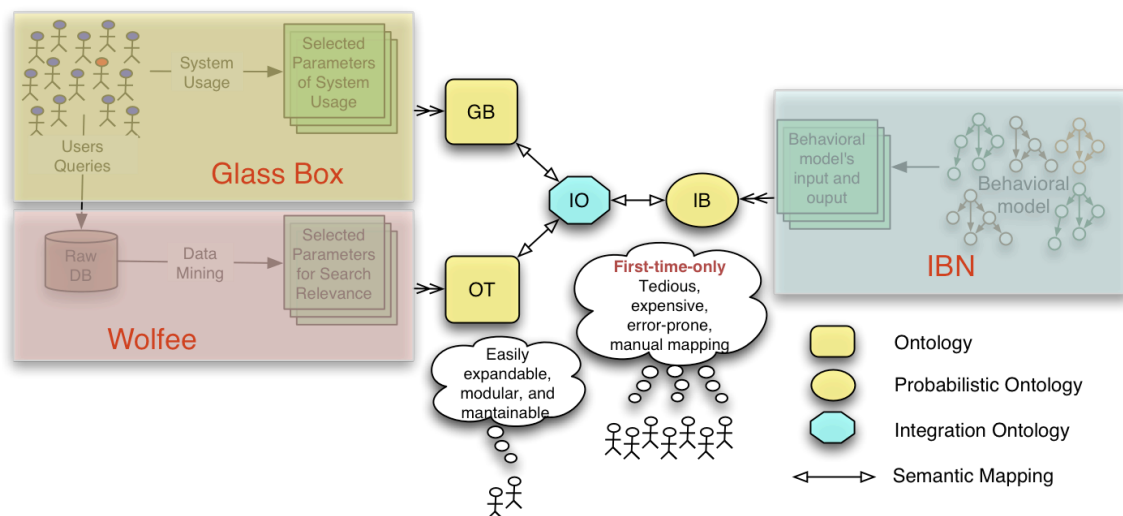


Figure 40. The Insider Threat Detection Process – Desired Process

It is important to note the nature of the IO. Contrary to most approaches in semantic mapping, the work in the DTB project is not towards a “merged”, “coarser”, bigger ontology containing domain information represented in GB, IB, or OT ontologies. Instead, IO is a domain-free, meta-ontology carrying only information about the semantic

mapping between each of the domain ontologies in the system. The structure, classes, and slots of the IO will not represent the domain in which the system being applied. Domain information will appear only in the instances of IO, which convey the actual relationships among the domain ontologies' concepts.

This approach for integration provides an elegant way of conveying semantic mapping information. Avoiding domain-related knowledge in the IO structure makes it much easier to maintain and to expand, as modifications in the ontologies being mapped will affect only the mappings (i.e. the instances of IO) and not the IO itself.

Some common problems arise from such a scheme, for example: how can we measure the commonality between any two concepts in different ontologies? Issues like that represent major constraints for any semantic mapping approach using deterministic frameworks, and it is where our research efforts will most likely provide breakthroughs in this area.

PR-OWL provides the necessary elements to overcome such limitations and is a suitable technology for building both the IB ontology (and any other ontology that has to represent probabilistic information) and the IO meta-ontology.

Therefore, PR-OWL has the potential to provide the DTB project with a modular, easily maintainable, expandable solution that will reduce the labor-intensive ontology mapping process for the initial setup only. After that, the probabilistic reasoning performed via the Integration Ontology will tremendously facilitate modifications or additions to the system.

## C.2 PR-OWL for Multi-Sensor Data Fusion: The Wise Pilot System

Bayesian Networks are much praised as a powerful tool for performing probabilistic inference, but they do have some limitations that impede their application to complex problems. To illustrate such issues, we present the Wise Pilot system (Costa, 1999), which analyzes a fighter aircraft sensors' information and assesses each eventual perceived track's relevant probability.

In a typical mission, here called a sortie, a fighter aircraft has to take off from a friendly aerodrome, perform a high-altitude flight over friendly territory, descend to a lower altitude preferably before being detected by the enemy's radar coverage, attack the mission's target(s) and egress home safely. Enemy's role is to detect the incoming fighter and deny its attack, using weapons like interceptors, AAA<sup>32</sup>, or missiles.

Behind the scenes lies a high-tech contest between the intruder fighter and enemy's forces, usually called electronic warfare. This contest can be compared to a hide-and-seek game, where the intruder fighter tries to stay out of enemies' electronic

---

<sup>32</sup> AAA is the acronym for Anti-Aircraft Artillery, which includes all gun-based weapons employed against airborne targets (aircraft, helicopters, cruise missiles, etc).

eyes (i.e. early warning radars, interceptor sensors, AAA radars, etc.) as long as he can. The ability of the intruder to hide from these hostile sensors will depend on tactics like low-level flight and reduction (or elimination) of communication and radar emissions. The first is intended to use the terrain as a mask against enemy's radar, while the latter avoids being discovered by the enemy's passive detectors.

However, flying into enemy territory means to be vulnerable to a wide array of threats, and for most of them awareness is the first requisite to improve the chances of surviving. To be aware, the pilot counts on the information provided by its own sensors, which can usually be grouped in two distinct types: passive and active. Sensors in the first group detect all transmissions and classify their respective sources; thus they do not need to make any transmissions by themselves. Sensors in the second group are those that transmit for a period of time and wait for a reply in order to obtain information.

Although passive sensors are a stealthy way of gathering information about the enemy, an obvious drawback is in the fact that their efficacy depends on whether the enemy is emitting or not. In addition, passive sensors like the RHAW do not provide a reliable measure of distance.

Active sensors, on the other hand, usually provide more accurate measurement. As a consequence, the decision to decrease uncertainty or detectability is a hard conflict to be solved, mainly during a high attention-demanding situation as a flight sortie. However, there are other issues regarding the use of active sensors. Among these is the management of the sensor's power, that is how to direct it (allocate it) for the many surrounding enemy's targets/aggressors.

The pilot should perform this allocation wisely, in order to achieve an optimal use of the aircraft's weapon systems (offensively and/or defensively). Here, the level of uncertainty will also influence the pilot's decisions.

Those decisions are not constrained to electronic warfare considerations. The pilot also has to deal with navigation issues, complex aircraft systems' monitoring, damage control, fuel consumption, and ultimately he still has to pilot his aircraft in a 540 knots near-the-ground flight.

In addition, modern aircraft and sophisticated defense systems have dramatically increased pilot's workload, particularly in the most critical phase of the mission, the attack.

The Wise Pilot system initially assesses the threat level of each track perceived by its sensors and then uses Bayesian Inference to provide the pilot with the best option available given the most up-to-date information on those tracks. Figure 41 summarizes the danger assessment process.

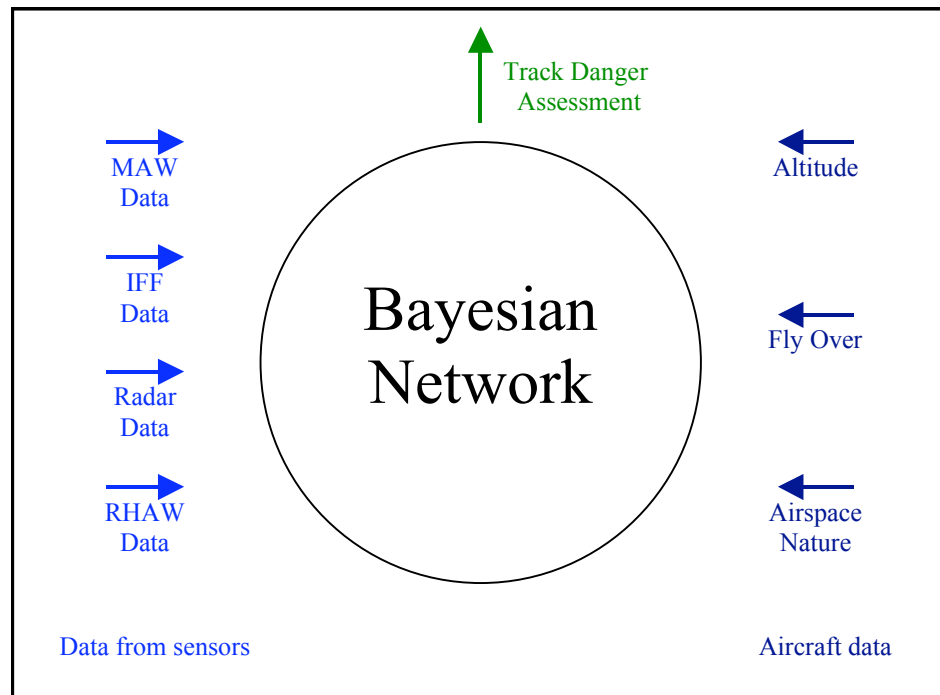


Figure 41. General Track Danger Assessment Scheme

A Bayesian Network, which is shown in more detail in Figure 42, receives data from the aircraft sensors related to one specific track (coming from the left in the picture) and from the aircraft's systems related to its own position, altitude, and other navigational details (coming from the right).

This information is then propagated inside the BN and will result in an assessment of the nature of that specific track and the potential danger it might represent.



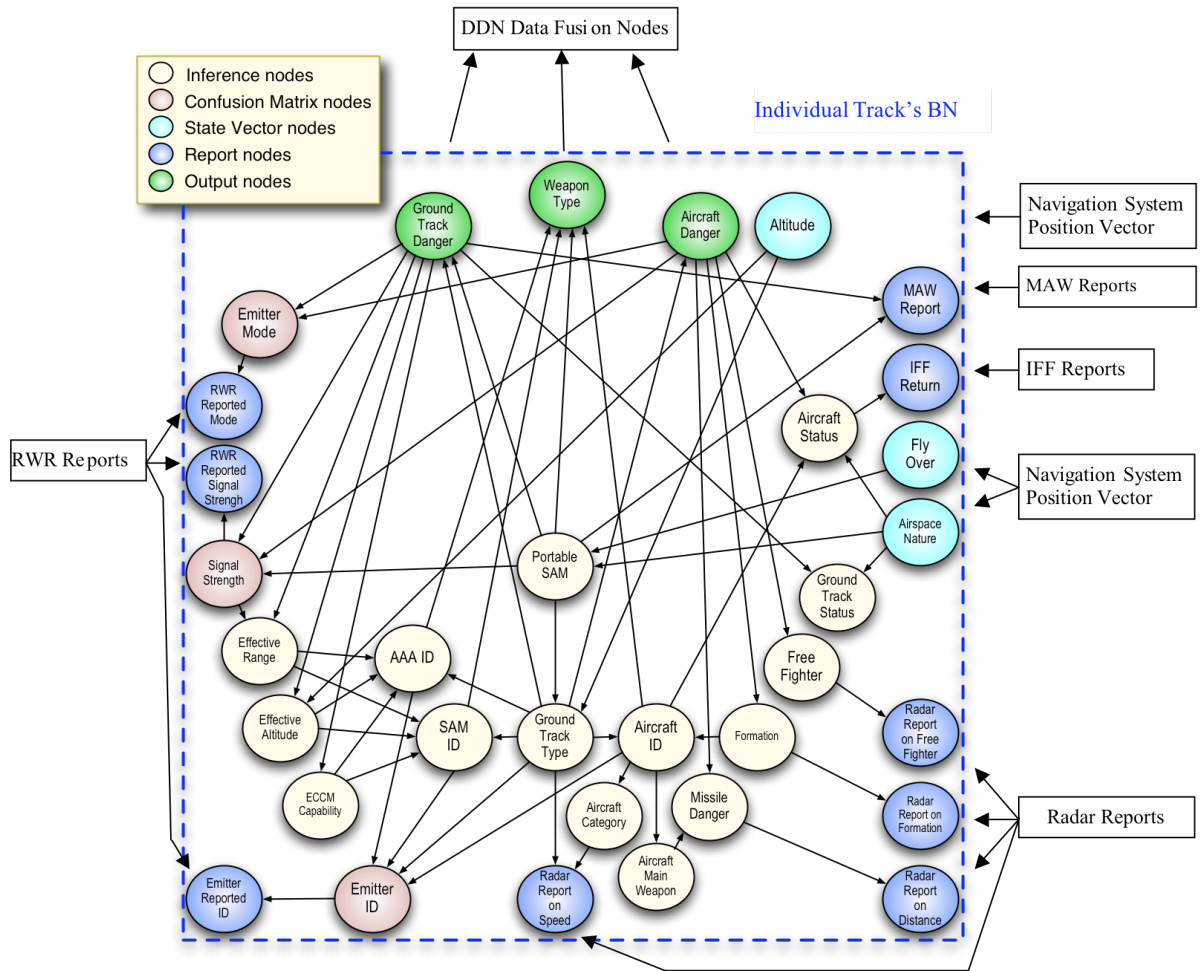


Figure 42. Individual Track's BN Information Exchange Scheme

Therefore, since each BN is responsible for the probabilistic assessment of each track, if we have  $n$  tracks we will have to have  $n$  Bayesian Networks for assessing its respective nature and potential danger. Figure 43 shows the general schema of the decision process to be performed after all data on tracks is received.

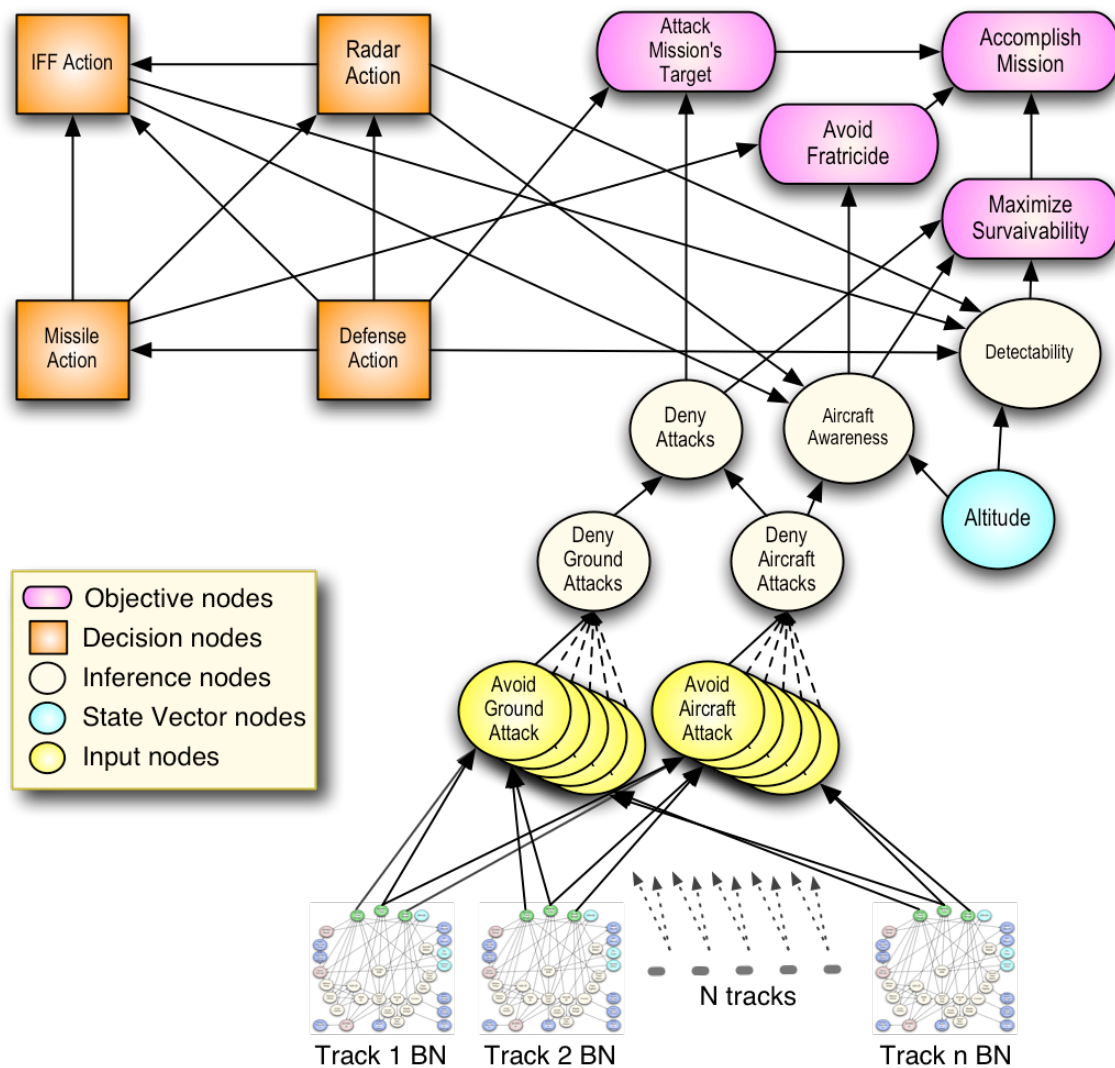


Figure 43. Wise Pilot system – general scheme

In the bottom of the picture we have the  $n$  Bayesian Networks related to the  $n$  tracks. The system works in discrete time, which means that at time  $t$  it will collect all information from the  $n$  tracks, use  $n$  BNs for assessing its nature and potential danger, and then evaluate what is the best combination of the four decisions it has to take (i.e. what is the best decision policy) for that specific situation given the objectives (i.e. attack target, avoid fratricide, and maximize survivability) and the most updated information available at time  $t$ .

The advantages of a Bayesian Inference system over deterministic rule-based systems for dynamic decision situations such as the fighter pilot problem have been discussed extensively in the literature (e.g. Costa, 1999). However, a major obstacle for implementation of this and similar systems is the lack of representational power of Bayesian Networks for dealing with the variable number of tracks for each time  $t$ .

As an example, Figure 44 shows how the system would look like in a given time  $t$  in which we have four tracks being perceived by the system.

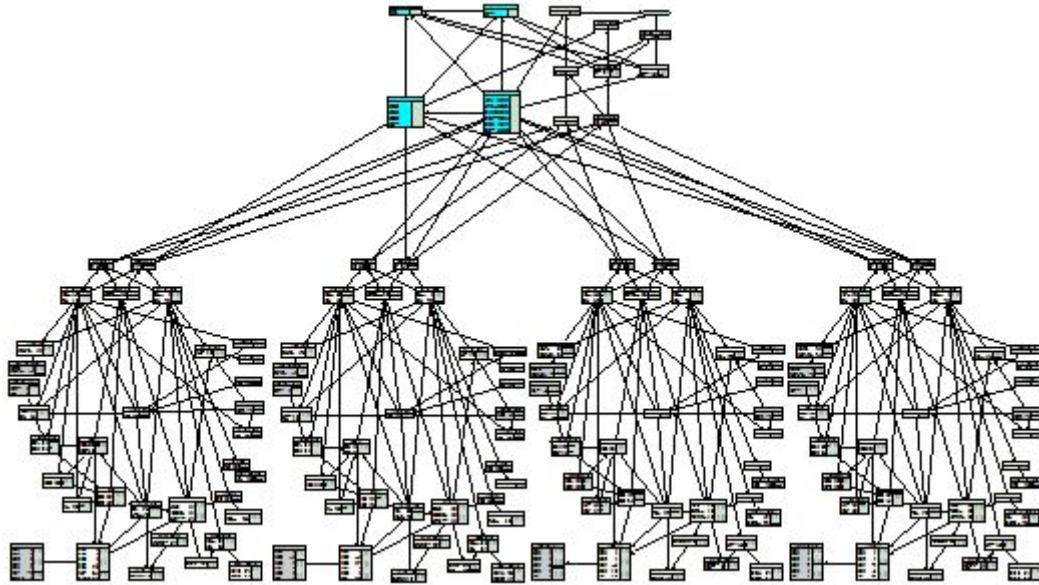


Figure 44. Wise Pilot with 4 Tracks

Now suppose that in time  $t+1$  (i.e. on the next system iteration) two new tracks are perceived by the system and one of the tracks from time  $t$  went away. In this new situation, the system depicted in Figure 44 is no longer valid, while the new configuration should look like the one illustrated in Figure 45.

In other words, in a highly dynamic environment like the one covered in this example, a Bayesian Inference system would have to be reconfigured almost for every iteration, greatly increasing the complexity of its implementation. Furthermore, there may be uncertainty about the correct configuration at any given iteration. Even more problematic is the situation in which we are unsure whether a sensor report indicates a real or spurious object, or whether two reports refer to the same or different subjects. In these cases, the number of instances of the track sub-network is uncertain.

Behind this limitation is the fact that Bayesian Networks have limited expressive power, while in situations like the one portrayed (and in many interesting situations of the real world as well) a more powerful representational formalism is desired.

More specifically, Bayesian Networks allow probability statements (i.e. propositions) over specific instances of a model, but do not support making general assertions over non-specific instances (e.g. statements about variables instead of unique instances). Thus, when facing problems like the fighter pilot's we need a language that combines the inferential power of Bayesian Networks with the representational power of first-order logic.

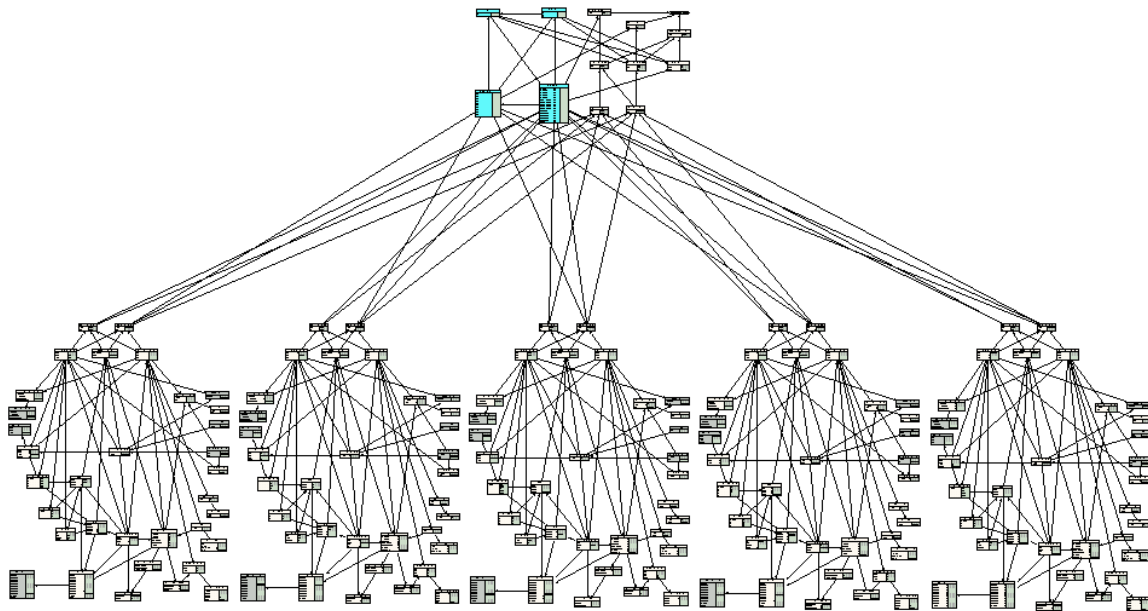


Figure 45. Wise Pilot with 5 Tracks

We have seen in Chapter 3 that MEBN logic provides such combination of Bayesian plausible reasoning and first-order logic expressiveness, and thus is a perfect match for the requirements. This is no surprise, since the suitability of MEBN logic for C3I decision systems was already pointed out in some recent research work (Costa *et al.*, 2005). However, for the very same reasons of the DTB project cited earlier in this appendix, MEBN alone would not guarantee that such system would be interoperable, easily maintainable and upgradeable.

Indeed, in order to realize the concept of Network Centric Warfare being sought by most modern armed forces (cf. Alberts *et al.*, 1999), massive investments must be made to achieve sensor interoperability and information sharing between combatant platforms in a tactical environment.

In the Wise Pilot case, building the system's multiple ontologies (e.g. ontologies on ground-based radar systems, airborne radars and respective platforms, interceptor aircraft and respective weapon systems, etc.) using PR-OWL would bring the intrinsic advantages of probabilistic ontologies, such as built-in ability to learn from previous engagements, and the possibility of improving maintainability, interoperability (among the system's ontologies and exterior ones as well), and expandability.

Of course, the above-cited advantages can be easily transposed to similar data fusion systems in the military domain and in civilian applications as well, clearly exposing the promising aspect of the technology outside the Semantic Web framework.

### Curriculum Vitae

Paulo Cesar G. da Costa was born in Rio de Janeiro on March 3, 1965, and is a Brazilian citizen. He graduated with honors in the Brazilian Air Force Academy in 1986 and started his career as a fighter pilot, having flown nearly 1.800 hours in fighter aircraft such as the Brazilian-Italian made AM-X. During this period, he pursued specialization in the Electronic Warfare field, attending courses in both Brazil and England, acting as Electronic Warfare Officer in the Brazilian Air Force's first AM-X Squadron, and then as an invited lecturer in most of the courses from the BAF's Electronic Warfare Center (CGEGAR). In 1995, he graduated first place out of 115 students, all in the rank of Captain, in the BAF's EAOAr, a major career course. In 1997, he moved to the USA and started his Master of Science degree in Systems Engineering at George Mason University, where he graduated in 1999 with GPA 4.0 and received the GMU's Academic Excellence Award. He also received the C3I Certificate from GMU in 1999. Back to Brazil in 2000, he served in the Air Force Chiefs of Staff, where he participated in many projects in the IT field. In 2003, he graduated with honors from the BAF's ECEMAR, another major career course, and returned to the USA to pursue his PhD degree in Information Technology at George Mason University.