BayesOWL: A Prototype System for Uncertainty in Semantic Web

Shenyong Zhang[1,2], Yi Sun[2], Yun Peng[2], Xiaopu Wang[1]
[1]Department of Astronomy and Applied Physics
University of Science and Technology of China, Hefei, Anhui 230026
[2]Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County, Baltimore, MD 21250
{syzhang, yisun1, ypeng}@umbc.edu, wxp@ustc.edu.cn

Abstract: Previously we have proposed a theoretical framework, called BayesOWL, to model uncertainty in semantic web ontologies based on Bayesian networks. In particular, we have developed a set of rules and algorithms to translate an OWL taxonomy into a BN. In this paper, we describe our implementation of BayesOWL framework together with examples of its use.

Keywords: semantic web, ontology, Bayesian networks, uncertainty reasoning, iterative proportional fitting procedure

1 Introduction

Semantic web as a formalism based on crisp description logic is known to be severely limited in representing and reasoning with uncertainty of the world it is meant to model. Various proposals have been made in the recent years to deal with uncertainty in semantic web, especially in the ontologies written in semantic web languages such as OWL and RDF [2,6]. These include BayesOWL, a probabilistic framework we proposed earlier that is aimed at automatically translating OWL ontologies to Bayesian networks (BNs). The details of this framework and its applications to ontology mapping can be found in our companion papers [3,4,5].

In this paper we focus on a prototype implementation of the BayesOWL framework. As can be seen in Figure 1 below, translation from an OWL ontology to a BN by the prototype system is done in two stages. The first stage is to construct the BN structure (a directed acyclic graph or DAG) from the input OWL ontology file and to initialize the conditional probability tables (CPTs) with default values. This is done by Taxonomy Parser (T-Parser for short) and BN structure constructor. The second stage is to incorporate user provided probabilistic information of concepts and inter-concept relations into the BN CPTs. This is done by Probability Parser (P-Parser for short) and CPT Constructor.

* This work was supported in part by NIST award 60NANB6D6206 and the China Scholarship Council (CSC).

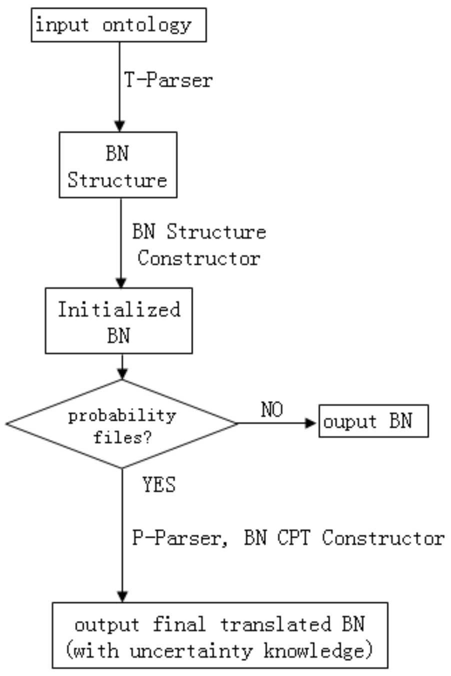


Figure 1. BayesOWL framework

For the benefit of readers, before presenting the prototype implementation, we briefly describe the BayesOWL framework in Section 2, especially the rules for structure translation and algorithms for incorporating probabilistic information into CPTs [3]. The detailed description of the prototype system and its implementation is then given in Section 3 together with some explanatory examples. Finally, Section 4 concludes with a summary and evaluation of this work and suggestions for future work.

2 From OWL ontology to Bayesian Networks

BayesOWL [3,4,5] is a framework that augments and supplements OWL for representing and reasoning with uncertainty based on Bayesian networks. This framework provides a set of rules and procedures for automatic translation of an OWL ontology into a BN structure (a DAG) and algorithms that incorporates probabilistic information about classes and inter class relations into the BN CPTs. The translated BN is shown to preserve the semantics of the original ontology and to be consistent with the probabilistic information; and it can support ontology reasoning, both within and across ontologies as Bayesian inferences.

2.1 Structure Translation

Given an OWL ontology that defines a concept taxonomy, BayesOWL uses the following rules to convert it into a BN DAG.

Rule 1: Concept Classes. Each defined concept class is mapped into a binary variable node, called concept node, in the translated BN with states "True" or "False" denoting whether a randomly selected individual belongs to this concept class or not.

Rule 2: Subclasses. If concept class *C* has a set of most specific super-concept classes C_i, a subnet is created in the translated BN with a converging connection from each C_i to *C* (see Figure 2).

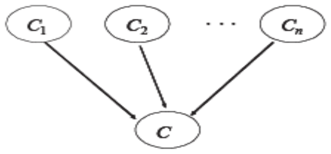


Figure 2. Subnet translated from "rdfs:subClassOf(*C*)"

Rule 3: Logical Relations. OWL uses five logical operators: intersection, union, complement, equivalent, and disjoint, to define logical relations between concept classes. For such a relation, a logic node (L-node) is created in the BN and is connected with the related concept nodes. For example, as shown in Figure 3, if concept class *C* is the intersection of concept classes C_i, a subnet is formed in the translated BN with one converging connection from each C_i to *C*, and one converging connection from *C* and each C_i to the logic node "LNodeIntersection". Logical relations defined using other logical operators can be similarly translated.

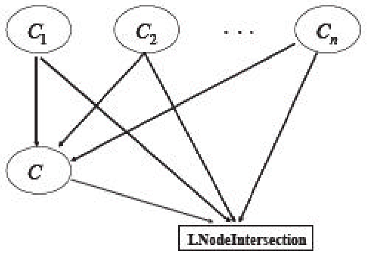


Figure 3. Subnet translated from "owl:intersectionof"

2.2 CPT Construction

The translated BN structure consists of two types of nodes: the logical nodes and the concept nodes. Next we describe how to construct CPTs for each of the two types of nodes.

2.2.1 CPT for L-Nodes

CPT for an L-Node can be completely determined by the logical relation it represents, that is, when its state is set to "True", the intended logical relation among its parents must hold. For example, if *C* is the intersection of C_1 and C_2, then the L-Node is "True" if and only if

$$c\,c_1c_2 \vee \overline{c}\,c_1\overline{c_2} \vee \overline{c}\,\overline{c_1}c_2 \vee \overline{c}\,\overline{c_1}\overline{c_2}\,.$$

This can be realized by the CPT shown in Table 1.
When CPTs of all L-Nodes have been filled, the states for L-Nodes are set to "True", so all logical relations hold.

Table 1. CPT for L-Node (intersection)

C	C_1	C_2	Intersection	
			True	False
True	True	True	1.0	0.0
True	True	False	0.0	1.0
True	False	True	0.0	1.0
True	False	False	0.0	1.0
False	True	True	0.0	1.0
False	True	False	1.0	0.0
False	False	True	1.0	0.0
False	False	False	1.0	0.0

2.2.2 Representing Probabilities in OWL

In many applications, probabilistic information of concept nodes and inter-concept relations such as prior probabilities for individual concepts and pair-wise conditional probabilities for subclass relations may be available for the given ontology. We require this information be encoded as OWL document. The encoding described in this subsection is a generalization of what we have proposed in [3, 5]. In this encoding we treat a probability as a kind of resource, and define several OWL classes to encode probabilities:

Class Variable: its instances denote variables (nodes) in the translated BN. A variable has a property called "hasClass", pointing to the concept class in the original ontology this variable is mapped from.

Class Proposition: its instances denote variable instantiations. A proposition has two properties: "hasVariable" and "haState", indicating the variable and the state the proposition is instantiating. Finally,

Class Probability: its instances denote individual probabilities. A probability has three properties: "hasProposition" (cardinality >= 1), "hasCondition" (cardinality >= 0) and "hasValue" (cardinality = 1).

Using these classes we can easily define marginal and conditional probabilities without ambiguity. This can be seen from the encoding of the conditional probability

$P(A = True \mid B = True, C = False) = 0.5$,

where A, B, C are BN nodes corresponding to the concepts ClassA, ClassB, and ClassC in the original ontology.

```
<owl:Variable rdf:ID="A">
    <hasClass>ClassA</hasClass>
</owl:Variable>
<owl:Variable rdf:ID="B">
    <hasClass>ClassB</hasClass>
</owl:Variable>
<owl:Variable rdf:ID="C">
    <hasClass>ClassC</hasClass>
</owl:Variable>
<owl:Proposition rdf:ID="a1">
    <hasVariable>A</hasVariable>
    <hasState>True</hasState>
</owl:Proposition>
<owl:Proposition rdf:ID="b1">
    <hasVariable>B</hasVariable>
    <hasState>True</hasState>
</owl:Proposition>
```

```
<owl:Proposition rdf:ID="c0">
    <hasVariable>C</hasVariable>
    <hasState>False</hasState>
</owl:Proposition>
<owl:Probability rdf:ID="P(a1|b1,c0)">
    <hasproposition>a1</hasproposition>
    <hasCondition>b1</hasCondition>
    <hasCondition>c0</hasCondition>
    <hasValue>0.5</hasValue>
</owl:Probability>
```

2.2.3 CPT for Concept Nodes

Let p_C be the set of all parent nodes of the concept node *C*. From the structure translation rules, all nodes in p_C are super-classes of *C*. Therefore, each entry in $P(C \mid p_C)$, the CPT of *C*, must have value zero if any of its parents is "False" for that entry. The only other entry in the table is the one in which all parents are "True". It is the probability distribution of this entry that needs to be determined.

If no probabilistic information is available, then CPT for each concept node is set for its default values. Among a number of alternatives, BayesOWL chooses the equal probability for the default. This can be seen from CPT for concept node C with parents A and B in Table 2 below.

Table 2. Default CPT for concept node C with parents A and B.

A	B	C	
		True	False
True	**True**	**0.5**	**0.5**
True	False	0.0	1.0
False	True	0.0	1.0
False	False	0.0	1.0

When probabilistic information is available, this information needs to be incorporated into the CPTs of the concept nodes. For this purpose we have developed several algorithms [8], all based on a mathematical procedure known as *iterative proportional fitting procedure* (IPFP) [1,7,12]. These algorithms take the probabilistic information as *constraints* and iteratively modify the joint distributions of all variables, using the constraints one at a time, until a convergence is reached in which all the constraints are satisfied by the resulting joint distribution.

The most widely available constraints for ontologies are those of marginals for individual concepts ($P(c)$) and pair-wise conditionals of subclass relations ($P(c|a)$). Note that each of these constraints involves variables within one CPT, and these constraints can be efficiently incorporated by one of our algorithms called D-IPFP, where D stands for *decomposed*. At each iteration, instead of modifying the joint distribution as general IPFP does, D-IPFP takes one constraint and only modifies one CPT which contains the variables of that constraint. The algorithm D-IPFP does as follows:

D-IPFP starts with initial joint distribution of translated BN, which is the product of CPTs of all concept nodes:

$$Q_{(0)} = P_{init}(X) = \Pi_{Xi\in X} P_{init}(X_i \mid p_i);$$

It then iterates over all constraints $R(x_i \mid L_i)$ and computes

$$Q_{(k)}(x_i \mid p_{x_i}) \cdot \frac{R(x_i \mid L_i)}{Q_{(k-1)}(x_i \mid L_i, LT)} \cdot a_{(k-1)}(p_{x_i});$$

where x_i is a concept node, L_i is the set of zero or more parents of x_i, $Q_{(k)}(x_i \mid p_{x_i})$ is the CPT that gets modified by constraint $R(x_i \mid L_i)$, LT denotes that all L-Nodes' states are set to be "True", and $a_{(k-1)}(p_{x_i})$ is the normalization factor which can be calculated by

$$a_{(k-1)}(p_{x_i}) = \frac{1}{\sum_{x_i} Q_{(k-1)}(x_i \mid p_{V_i}) \cdot R(x_i \mid L_i) / Q_{(k-1)}(x_i \mid L_i, LT)}.$$

3 BayesOWL Prototype System

Our prototype implementation of BayesOWL is written in Java. It takes two inputs: (1) an OWL file that defines the ontology that is to be translated into a BN; and (2) an OWL file of the probabilistic information of concepts and inter-concept relations encoded in the way as described in Subsection 2.2.2. Note that the current implementation only translates the terminological taxonomy of the given ontology into a BN, and the probabilistic information is restricted to only those of marginals for individual concepts and pair-wise conditionals for subclass relations.

The translated BN is written in the format of Netica, a Bayesian network development software system from Norsys Software Corporation [17].

3.1 System Architecture

BayesOWL prototype system contains a series of APIs, a graphical user interface (GUI) and related documentations. The system architecture of BayesOWL is shown in Figure 4.

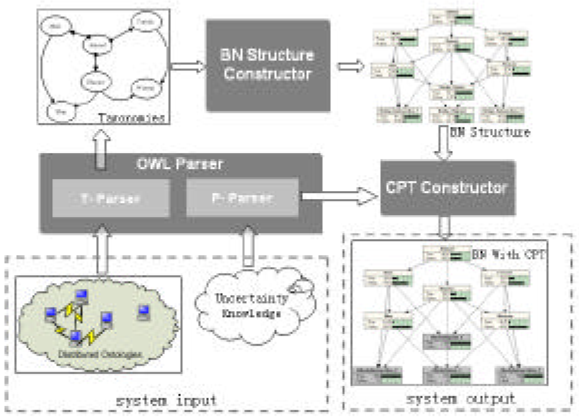


Figure 4. BayesOWL system architecture

In the implementation, Jena API is used in T-Parser to parse OWL ontology file. Jena is an open source Java framework for semantic web applications. It is grown out of work of the HP Labs Semantic Web Program and provides OWL APIs (see [16] for details). T-Parser outputs the list of all concept nodes, one for each concept class defined in the ontology file, together with defined super-classes for each concept. It also creates L-Nodes, one for each defined logical relation.

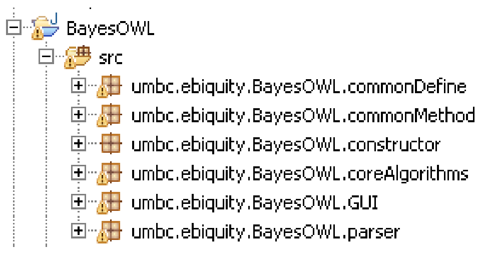
The BN Constructor takes the output from T-Parser and translates the nodes into the BN structure (the DAG) according to the structure translation rules given in Section 2.1. It also initializes the CPTs for all nodes as described in Subsections 2.2.1 (for L-Nodes) and 2.2.3 (for concept nodes).

Component P-Parser is built to parse probability files and extract the encoded probabilities into a specific format. These probabilities are then taken by CPT Constructor as input constraints for D-IPFP algorithm to modify the CPTs of the concept nodes.

3.2 BayesOWL API

Figure 5 shows BayesOWL APIs, which are contained in several Java packages. The package "commonDefine" contains classes defining data structure such as joint probability table etc. The package "commonMethod" contains a list of operations for the defined data structures. IPFP based algorithms are packed in the package "coreAlgorithms". Both T-Parser and P-Parser are defined in package "parser". The package "constructor" consists of BN structure constructor and CPT constructor. Finally, the package "GUI" implements the system's

Graphical User Interface. All of these packages work together to complete the Ontology-to-BN translation. Each of these packages can also be used separately.



```
BayesOWL
  src
    umbc.ebiquity.BayesOWL.commonDefine
    umbc.ebiquity.BayesOWL.commonMethod
    umbc.ebiquity.BayesOWL.constructor
    umbc.ebiquity.BayesOWL.coreAlgorithms
    umbc.ebiquity.BayesOWL.GUI
    umbc.ebiquity.BayesOWL.parser
```

Figure 5. BayesOWL API

3.3 BayesOWL GUI

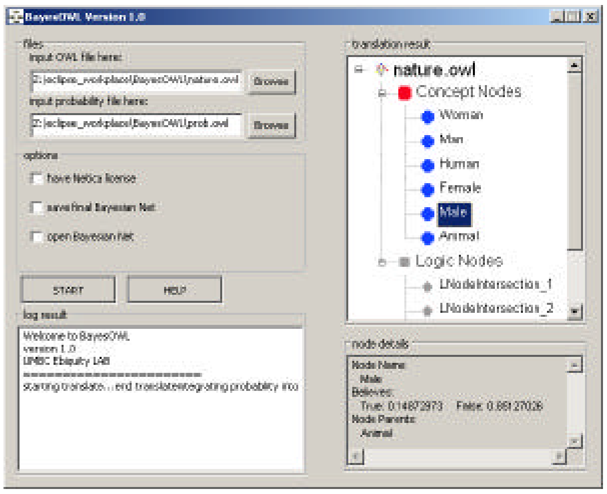
The system GUI is given in Figure 6. The layout is divided into several areas:

- *File input area*, which is used to input OWL ontology files and probability files;
- *Options area*, which is designed for optional operations such as requesting Netica license for large BN, the location the result BN is to be saved, and if you want to open and view the result BN when it is generated;
- *Log area*, which is built for showing the running status;
- *Result BN area*, which shows the translated BN in a tree structure; and
- *Node detail area*, which gives node details, including its prior beliefs and its parents, when a node is selected in result BN area.

The BayesOWL GUI is executable. After the input ontology and probability files are specified, the "start" button starts the translation, the result BN will be generated and saved, and the network structure is shown in the translation result area.

3.4 Examples

We demonstrate the validity of our approach and the implementation by a simple example ontology called "nature", taken from [5]. This ontology defines the following six concept classes and several logical relations among these concepts:



input OWL file here: D:\eclipse_workplace\BayesOWL\nature.owl

input probability file here: D:\eclipse_workplace\BayesOWL\prob.owl

☐ have Netica license

☐ save final Bayesian Net

☐ open Bayesian Net

```
Welcome to BayesOWL
version 1.0
UMBC Ebiquity LAB
==============================
starting translate....end translateintegrating probability into
```

```
nature.owl
  Concept Nodes
    Woman
    Man
    Human
    Female
    Male
    Animal
  Logic Nodes
    LNodeIntersection_1
    LNodeIntersection_2
```

```
Node Name:
  Male
Believes:
  True: 0.14872973    False: 0.85127026
Node Parents:
  Animal
```

Figure 6. GUI of BayesOWL

- "Animal" is a primitive concept class;
- "Male", "Female" and "Human" are subclasses of "Animal";
- "Man" and "Woman" are two subclasses of "Human";
- "Male" and "Female" are disjoint with each other;
- "Man" is an intersection of "Human" and "Male";
- "Woman" is an intersection of "Human" and "Female";
- "Human" is the union of "Man" and "Woman".

Figure 7 gives the BN structure translated from this ontology. It contains six concept nodes, one per each concept class, together with directed links for the defined subclass relations. The BN also contains four L-Nodes for the four defined logical relations, together with the proper links as dictated by the structure translation rules for these logical relations. All nodes' CPTs are initialized using rules discussed in Subsections 2.2.1 and 2.2.3.

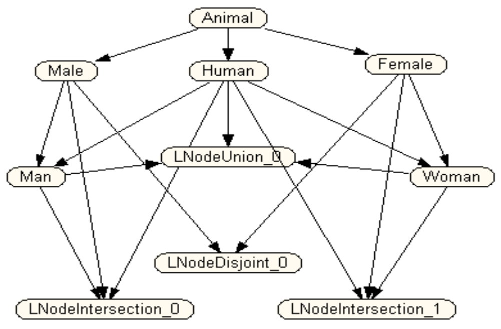


Figure 7. Translated BN structure from the "nature" ontology

Figure 8 shows the final BN after CPTs are modified by the following probabilistic constraints:

- P(Animal) = 0.5
- P(Male|Animal) = 0.5
- P(Female|Animal) = 0.48
- P(Human|Animal) = 0.1
- P(Man|Human) = 0.49
- P(Woman|Human) = 0.51

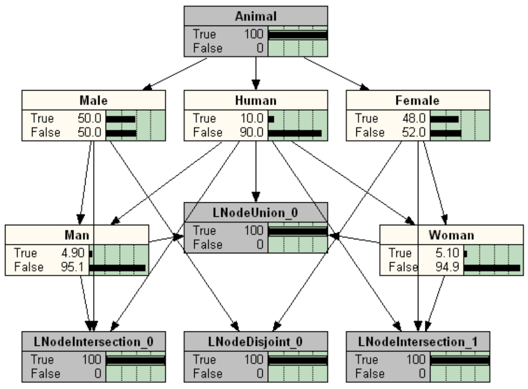


Figure 8. Translated BN of the "nature" ontology

The final CPTs for the six concept nodes are given in Figure 9 below.

Animal	
True	False
0.92752	0.07248

Animal	Human	
	True	False
True	0.18773	0.81227
False	0.0	1.0

Animal	Female	
	True	False
True	0.95469	0.04531
False	0.0	1.0

Animal	Male	
	True	False
True	0.95677	0.04323
False	0.0	1.0

Male	Human	Man	
		True	False
True	True	0.47049	0.52951
True	False	0.0	1.0
False	True	0.0	1.0
False	False	0.0	1.0

Female	Human	Woman	
		True	False
True	True	0.51433	0.48567
True	False	0.0	1.0
False	True	0.0	1.0
False	False	0.0	1.0

Figure 9. Final CPTs of BN for "nature" ontology

In Figure 8 we can see that all L-Nodes are set to "True", and that, when "Animal" is set to "True", probabilities of "Male", "Female" and "Human" are all consistent with the given probabilistic constraints.

We have also experimented BayesOWL with several large ontologies. One example is the ontology "SWRC.owl" for modeling entities of research communities [11]. The result BN for SWRC.owl contains 125 variables, including 70 mapped from concept classes and 55 L-Nodes.

3.5 Applications

The translated BN can support common ontological reasoning tasks as probabilistic inferences in the subspace of *LT* (i.e., all L-Nodes are set to "True"). One of such tasks is *Concept Satisfiability*: deciding whether a concept class represented by a description *x* is null. This can be done by calculating if $P(e|LT)=0$. Another task is *Concept Overlapping*: deciding if and to what extent a description *x* overlaps with a concept *C*. This can be measured by $P(c\,|\,x, LT)$, which can be computed by applying general BN belief update algorithms.

Another reasoning task with wide applications is to find the concept *C* defined in an ontology that is semantically most similar to a description *x*. Without the probabilistic extension, this task is often accomplished by finding the most specific subsumer of *x*. With BayesOWL, this can be done by finding the concept with the maximum similarity score in some similarity measure such as Jaccard coefficient

$$J(c,x) = P(c \wedge x\,|\,CT)/P(c \vee x\,|\,CT).$$

For example, let $x = \neg Male \wedge Animal$ in our "nature" ontology. The most specific subsumer found by the semantic web reasoner Racer is "Animal". But using BayesOWL the most similar concept is "Female" with *J*(*x*, *Female*) = 0.9593, which is certainly a more reasonable answer than "Animal".

BayesOWL also supports reasoning for uncertainty descriptions. Continuing our example, suppose now *x* is described as belonging to class "Male" with probability 0.1 and to "Animal" with probability 0.7. Then *J*(*c, x*) can be computed by inputting these probabilities as virtual evidence to the BN [9]. Class "Female" remains the most similar concept to *x*, but its similarity score *J*(*x*, *Female*) now decreases to 0.5753.

4 Conclusions and Future Work

In this paper, we described a prototype implementation for BayesOWL, a probabilistic framework for uncertainty in semantic web ontologies based on Bayesian networks. The implementation contains a series of APIs, including algorithms for structure translation of an OWL terminological taxonomy to a BN DAG and for incorporating probabilistic constraints into the BN CPTs. A graphical user interface is also implemented for facilitating the execution of the system. Experiments show the system runs well on OWL taxonomies of different size.

As a software tool, the BayesOWL system can be used by researchers and practitioners on ontology engineering such as domain modeling, ontology reasoning and ontology concept mapping [8].

Further research is required along several directions. One is to extend the theoretical framework from taxonomies to full OWL ontologies that include properties and data types. Another is to deal with probabilistic constraints that are inconsistent with each other, a situation often occurring with real applications. Several proposals have been made to modify a joint distribution with inconsistent constraints [12,13], including our own [14]. It is our plan to adopt some of these methods into the CPT constructor in the next release of BayesOWL. Finally, it is desirable to automatically learn the probabilistic constraints from existing resources such as the relevant websites and/or text corpora when such constraints are not available from the domain experts or the ontology designers.

5 References

[1] Csiszar, I., "I-divergence Geometry of Probability Distributions and Minimization Problems", *The Annuals of Probability*, 3(1): 146-158, 1975.

[2] Costa, P., Laskey, K. B., and Laskey, K. J., "PR-OWL: A Bayesian ontology language for the semantic web", in *Proceedings of Workshop on Uncertainty Reasoning for the Semantic Web (URSW) at the 4th International Semantic Web Conference (ISWC)*, Galway, Ireland, November 2005.

[3] Ding, Z., Peng, Y., "A Probabilistic Extension to Ontology Language OWL", In *Proceedings of the 37th Hawaii International Conference on System Sciences*, Big Island, HI, 2004.

[4] Ding, Z., Peng, Y., Pan, R., "A Bayesian Approach to Uncertainty Modeling in OWL Ontology", In *Proceedings of 2004 International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA)*, Luxembourg-Kirchberg, Luxembourg, 2004.

[5] Ding, Z., Peng, Y. and Pan, R., "BayesOWL: Uncertainty Modeling in Semantic Web Ontologies", in *Soft Computing in Ontologies and Semantic Web*, Springer-Verlag, March 2006.

[6] Fukushige, Y., "Representing Probabilistic Knowledge in the Semantic Web", Position paper for *the W3C Workshop on Semantic Web for Life Sciences*, Cambridge, MA, USA, October 2004.

[7] Kruithof, R., Telefoonverkeersrekening, De Ingenieur 52, E15-E25, 1937.

[8] Pan, R., Ding, Z., Yu, Y. and Peng, Y., "A Bayesian Network Approach to Ontology Mapping", in *Proceedings of the Fourth International Semantic Web Conference (ISWC2005)*, Galway, Ireland, November 6-10, 2005.

[9] Pearl, J., "Jeffery's Rule, Passage of Experience, and Neo-Bayesianism", In *Knowledge Representation and Defeasible Reasoning*, H.E. Kyburg Jr. et al (eds), 245-265, Kluwer Academic Publishers, 1990.

[10] Peng, Y. and Ding, Z., "Modifying Bayesian Networks by Probability Constraints", in *Proceedings of 21st Conference on Uncertainty in Artificial Intelligence (UAI-2005)*, Edinburgh, Scotland, July 26-29, 2005.

[11] Sure, Y., et. al, "The SWRC Ontology - Semantic Web for Research Communities", In *Proceedings of the 12th Portuguese Conference on Artificial Intelligence (EPIA 2005)*, Springer, Covilha, Portugal, December 2005.

[12] Vomlel, J., "Methods of Probabilistic Knowledge Integration", PhD thesis, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University, 1999.

[13] Vomlel, J., "Integrating Inconsistent Data in a Probabilistic Model", *Journal of Applied NonClassical Logics*, pp. 1-20, 2003.

[14] Zhang, S. and Peng, Y., "An Efficient Method for Probabilistic Knowledge Integration", in *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-2008)*, Dayton, Ohio, Nov. 3-5, 2008.

[15] http://www.w3.org/2004/OWL

[16] http://jena.sourceforge.net/

[17] http://www.norsys.com/

[18] http://ontoware.org/projects/swrc/