UMP-ST plug-in: a tool for documenting, maintaining, and evolving probabilistic ontologies

Rommel N. Carvalho[1], Marcelo Ladeira[2], Rafael M. de Souza[2], Shou Matsumoto[2], Henrique A. da Rocha[1], and Gilson L. Mendes[1]

[1] Department of Strategic Information
Brazilian Office of the Comptroller General
SAS, Quadra 01, Bloco A, Edifício Darcy Ribeiro
Brasília, Distrito Federal, Brazil
{rommel.carvalho,henrique.rocha,liborio}@cgu.gov.br
http://www.cgu.gov.br

[2] Department of Computer Science
University of Brasília
Campus Universitário Darcy Ribeiro
Brasília, Distrito Federal, Brazil
mladeira@unb.br, {rafaelmezzomo,cardialfly}@gmail.com
http://www.cic.unb.br

Abstract. Although several languages have been proposed for dealing with uncertainty in the Semantic Web (SW), almost no support has been given to ontological engineers on how to create such probabilistic ontologies (PO). This task of modeling POs has proven to be extremely difficult and hard to replicate. This paper presents the first tool in the world to implement a process which guides users in modeling POs, the Uncertainty Modeling Process for Semantic Technologies (UMP-ST). The tool solves three main problems: the complexity in creating POs; the difficulty in maintaining and evolving existing POs; and the lack of a centralized tool for documenting POs. Besides presenting the tool, which is implemented as a plug-in for UnBBayes, this papers also presents how the UMP-ST plug-in could have been used to build the Probabilistic Ontology for Procurement Fraud Detection and Prevention in Brazil, a proof-of-concept use case created as part of a research project at the Brazilian Office of the General Comptroller (CGU).

Keywords: Uncertainty Modeling Process, Semantic Web, UMP-ST, POMC, Probabilistic Ontology, Fraud Detection, MEBN, UnBBayes

1 Introduction

In the last decade there has been a significant increase in formalisms that integrate uncertainty representation into ontology languages. This has given birth to several new languages like: PR-OWL [5–7], PR-OWL 2 [4, 3], OntoBayes [20], BayesOWL [8], and probabilistic extensions of SHIF(**D**) and SHOIN(**D**) [15].

However, the increase of expressive power these languages have provided did not come without its drawbacks. In order to express more, the user is also expected to deal with more complex representations. This increase in complexity has been a major obstacle to making these languages more popular and used more often in real world problems.

While there is a robust literature on ontology engineering [1, 10] and knowledge engineering for Bayesian networks [14, 12], the literature contains little guidance on how to model a probabilistic ontology.

To fill the gap, Carvalho [4] proposed the Uncertainty Modeling Process for Semantic Technologies (UMP-ST), which describes the main tasks involved in creating probabilistic ontologies.

Nevertheless, the UMP-ST is only a guideline for ontology designers. In this paper we present the UMP-ST plug-in for UnBBayes. This plug-in has the objective of overcoming three main problems:

1. the complexity in creating probabilistic ontologies;
2. the difficulty in maintaining and evolving existing probabilistic ontologies; and
3. the lack of a centralized tool for documenting probabilistic ontologies.

This paper is organized as follows. Section 2 introduces the UMP-ST process and the Probabilistic Ontology Modeling Cycle (POMC). Section 3 presents UnBBayes and its plug-in framework. Then, Section 4 describes UMP-ST plug-in, which is the main contribution of this paper. Section 5 illustrates how this tool could have been used to create a probabilistic ontology for procurement fraud detection and prevention. Finally, Section 6 presents some concluding remarks.

2 UMP-ST

The Uncertainty Modeling Process for Semantic Technologies (UMP-ST) consists of four major disciplines: Requirements, Analysis & Design, Implementation, and Test.

Figure 1 depicts the intensity of each discipline during the UMP-ST is iterative and incremental. The basic idea behind iterative enhancement is to model the domain incrementally, allowing the modeler to take advantage of what is learned during earlier iterations of the model. Learning comes from discovering new rules, entities, and relations that were not obvious previously. Some times it is possible to test some of the rules defined during the Analysis & Design stage even before having implemented the ontology. This is usually done by creating simple probabilistic models to evaluate whether the model will behave as expected before creating the more complex first-order probabilistic models. That is why some testing occurs during the first iteration (I1) of the Inception phase, prior to the start of the implementation phase.

Figure 2 presents the Probabilistic Ontology Modeling Cycle (POMC). This cycle depicts the major outputs from each discipline and the natural order in which the outputs are produced. Unlike the waterfall model [17], the POMC

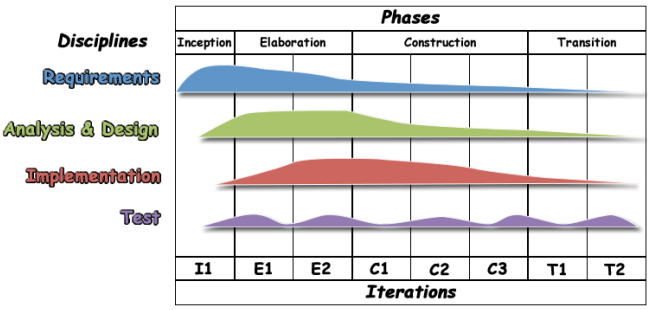


Fig. 1. Uncertainty Modeling Process for Semantic Technologies (UMP-ST).

cycles through the steps iteratively, using what is learned in one iteration to improve the result of the next. The arrows reflect the typical progression, but are not intended as hard constraints. Indeed, it is possible to have interactions between any pair of disciplines. For instance, it is not uncommon to discover a problem in the rules defined in the Analysis & Design discipline during the activities in the Test discipline. As a result, the engineer might go directly from Test to Analysis & Design in order to correct the problem.

Fig. 2. Probabilistic Ontology Modeling Cycle (POMC) - Requirements in blue, Analysis & Design in green, Implementation in red, and Test in purple.

In Figure 2 the Requirements discipline (blue circle) defines the goals that should be achieved by reasoning with the semantics provided by our model. The Analysis & Design discipline describes classes of entities, their attributes, how they relate, and what rules apply to them in our domain (green circles). This definition is independent of the language used to implement the model. The Implementation discipline maps our design to a specific language that allows uncertainty in semantic technologies (ST). For our case study, the mapping is to PR-OWL (red circles). Finally, the Test discipline is responsible for evaluating whether the model developed during the Implementation discipline is behaving as expected from the rules defined during Analysis & Design and whether they achieve the goals elicited during the Requirements discipline (purple circle). As noted previously, it is a good idea to test some rules and assumptions even before the implementation. This is a crucial step to mitigate risk by identifying problems before wasting time in developing an inappropriate complex model.

An important aspect of the UMP-ST process is defining traceability of requirements. Gotel and Finkelstein [11] define requirements traceability as:

> Requirements traceability refers to the ability to describe and follow the life of a requirement, in both forward and backward directions.

To provide traceability, requirements should be arranged in a specification tree, so that each requirement is linked to its "parent" requirement. In our procurement model, each item of evidence is linked to a query it supports, which in turn is linked to its higher level goal. This linkage supports requirements traceability.

In addition to the hierarchical decomposition of the specification tree, requirements should also be linked to work products of other disciplines, such as the rules in the Analysis & Design discipline, probability distributions defined in the Implementation discipline, and goals, queries, and evidence elicited in the Requirements discipline. These links provide traceability that is essential to validation and management of change.

This kind of link between work products of different disciplines is typically done via a Requirements Traceability Matrix (RTM) [19, 18]. Although useful and very important to guarantee the goals are met, the RTM is extremely hard to keep track without a proper tool. Therefore, this was a crucial feature that we incorporated into the UMP-ST plug-in.

3 UnBBayes plug-in Architecture

UnBBayes is an open-source Java™ application developed by the Artificial Intelligence Group from the Computer Science Department at the University of Brasilia in Brazil that provides a framework for building probabilistic graphical models and performing plausible reasoning. It features a graphical user interface (GUI), an application programming interface (API), as well as plug-in support for unforeseen extensions. It offers a comprehensive programming model that supports the exploitation of probabilistic reasoning and intrinsically provides a

high degree of scalability, thus presenting a means of developing AI systems on the fly [13, 16].

Unlike APIs, plug-ins offer a means to run new code inside the UnBBayes' runtime environment. A plug-in is a program that interacts with a host application (a core) to provide a given function (usually very specific) "on demand". The binding between a plug-in and a core application usually happens at loading time (when the application starts up) or at runtime.

In UnBBayes, a plug-in is implemented as a folder, a *ZIP* or a *JAR* file containing the following elements: (a) **a plug-in descriptor file**[3] (a XML file containing meta-data about the plug-in itself), (b) **classes** (the Java program itself - it can be a set of "*.class*" files or a packaged *JAR* file), and (c) **resources** (*e.g.* images, icons, message files, mark-up text).

UnBBayes currently relies on Java plug-in Framework (JPF) version 1.5.1 to provide a flexible plug-in environment. JPF is an open source plug-in infrastructure framework for building scalable Java projects, providing a runtime engine that can dynamically discover and load plug-ins on-the-fly. The activation process (*i.e.* the class loading process) is done in a lazy manner, so plug-in classes are loaded into memory only when they are needed.

One specific type of plug-in that can be added to UnBBayes is the module plug-in. Module plug-ins provide a means to create a relatively self-sufficient feature in UnBBayes (*e.g.* new formalisms or completely new applications). In UnBBayes vocabulary, *modules* are basically new internal frames that are initialized when tool bars or menu buttons are activated. Those internal frames do not need to be always visible, so one can create modules that add new functionalities to the application without displaying any actual "internal" frame (wizards or pop-ups can be emulated this way). The UMP-ST tool presented in this paper is a completely new application, since it was implemented as a module plug-in.

Figure 3 illustrates the main classes of a module plug-in. `UnBBayesModule` is the most important class of a module and it is an internal frame (thus, it is a subclass of *swing* `JInternalFrame`). Classes implementing `IPersistenceAwareWindow` are GUI classes containing a reference to an I/O class, and because `UnBBayesModule` implements `IPersistenceAwareWindow`, a module should be aware of what kind of files it can handle (so that UnBBayes can consistently delegate I/O requests to the right modules). `NewModuleplug-in` and `NewModuleplug-inBuilder` are just placeholders representing classes that should be provided by plug-ins. The builder is necessary only if `NewModuleplug-in` does not provide a default constructor with no parameters. For more information on UnBBayes plug-in framework see [16].

4 UMP-ST plug-in

As seen in Section 2, the UMP-ST process consists of four major disciplines: Requirements, Analysis & Design, Implementation, and Test. Nevertheless, the

[3] A plug-in descriptor file is both the main and the minimal content of a UnBBayes plug-in, thus one can create a plug-in composed only by a sole descriptor file.

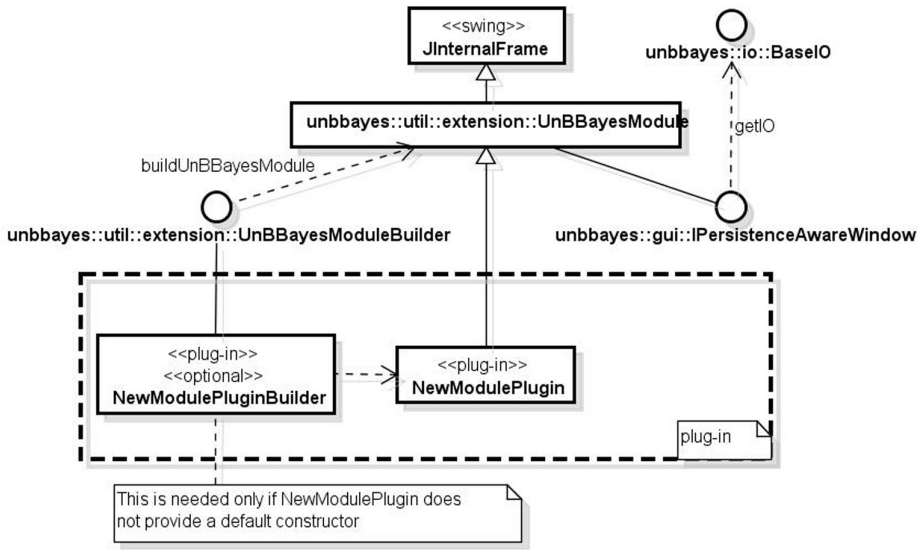


Fig. 3. Class diagram of classes that must be extended to create a module plug-in.

UMP-ST plug-in focuses only on the Requirements and Analysis & Design disciplines, since they are the only language independent disciplines. Moreover, as explained in Section 1, the objective of the UMP-ST plug-in is overcoming three main problems:

1. the complexity in creating probabilistic ontologies;
2. the difficulty in maintaining and evolving existing probabilistic ontologies; and
3. the lack of a centralized tool for documenting probabilistic ontologies.

The UMP-ST plug-in is a almost like a wizard tool that guides the user in each and every step of the Requirements and Analysis & Design disciplines. This involves the definition of the goals that should be achieved by the probabilistic ontology (PO) as well as the queries that should be answered by the PO in order to achieve that goal and the evidence needed in order to answer these queries. Only then the user is allowed to move to the next phase of the process which is defining the entities, then the rules, and finally the groups related to the defined goals, queries, and evidence (see Figure 2).

Respecting this order of steps defined in the process allows the tool to incorporate an important aspect which is traceability. In every step of the way, the user is required to associate which working product previously defined requires the definition of this new element. For instance, when defining a new query, the user has to say which goal that query helps achieve. We call this feature backtracking. This feature allows, for instance, the user to identify which goals are being achieved by the implementation of a specific group. This feature provides an easy and friendly way of maintaining the RTM matrix, defined previously.

The step by step guidance provided by the tool allows the user to overcome the complexity in creating POs (first problem). Moreover, the plug-in also solves the third problem, since all the documentation related to the PO being designed is centralized in the tool and can be saved for future use.

Finally, the difficulty in maintaining and evolving existing POs (second problem) is addressed mainly by the traceability feature. When editing any element (e.g., a goal, an entity, a rule, etc), two panels are always present. On the one hand, the back-tracking panel shows every element from previous steps of the process associated with the element being edited. On the other hand, the forward-tracking panel shows every element created in the following steps of the process associated with the element being edited. This provides a constant attention to where and what your changes might impact, which facilitates maintainability and evolution of existing POs.

Figure 4 presents the panel for editing entities with some of the main features of the UMP-ST plug-in.

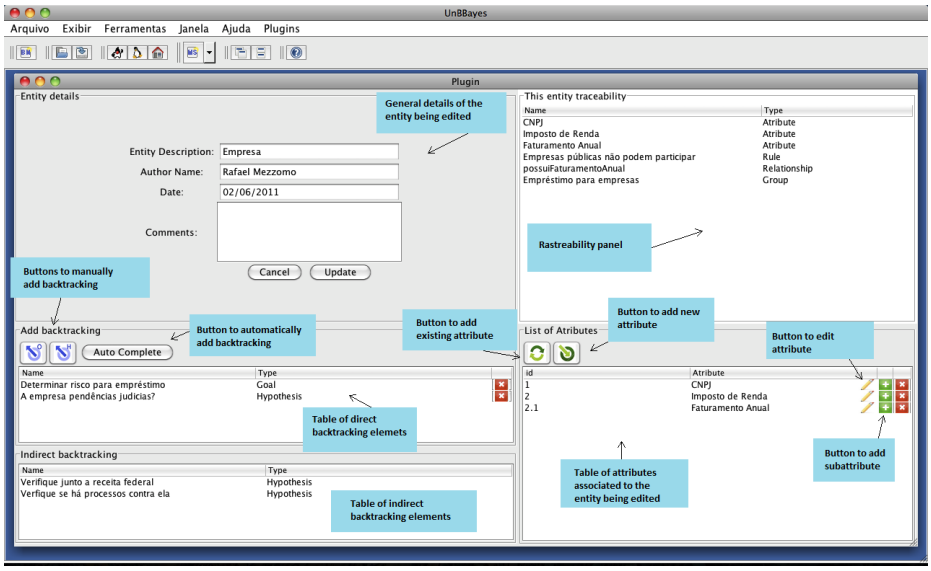


Fig. 4. Panel for editing an entity with a few features highlighted.

The UMP-ST tool was implemented as a module plug-in in UnBBayes. The UMP-ST plug-in is mostly structured in a *Model-View-Controller* (MVC[4]) de-

[4] A MVC design isolates logic and data from the user interface, by separating the components into three independent categories: *Model* (data and operations), *View* (user interface) and *Controller* (mostly, a mediator, scheduler, or moderator of other classes) [2].

sign pattern[5], which explicitly separates the program's elements into three distinct roles, in order to provide *separation of concern* (*i.e.* the software is separated into three different set of classes with minimum overlap of functionality). The View is implemented by the `umpst.GUI` package, the Controller by the `umpst.Controller` package, and the Model by the `umpst.IO` and `umpst.Model` packages. For more details, see [16].

5 Use Case

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

This section presents how the UMP-ST plug-in could have been used to build the Probabilistic Ontology for Procurement Fraud Detection and Prevention in Brazil, an use case presented by Carvalho [4]. Although Carvalho [4] has followed the UMP-ST process, there was no tool at the time to help create the corresponding documentation. The focus of this section is to show how this modeling process could benefit from the UMP-ST plug-in[6].

As explained in Section 2, the objective of the Requirements discipline is to define the objectives that should be achieved by representing and reasoning with a computable representation of domain semantics. For this discipline, it is important to define the questions that the model is expected to answer, *i.e.*, the queries to be posed to the system being designed. For each question, a set of information items that might help answer the question (evidence) should be defined.

One of the goals presented in [4] with its respective queries/evidences is:

1. *Goal*: Identify whether the committee of a given procurement should be changed.
   (a) *Query*: Is there any member of committee who does not have a clean history?
      i. *Evidence*: Committee member has criminal history;
      ii. *Evidence*: Committee member has been subject to administrative investigation.
   (b) *Query*: Is there any relation between members of the committee and the enterprises that participated in previous procurements?
      i. *Evidence*: Member and responsible person of an enterprise are relatives (mother, father, brother, or sister);

[5] *Design patterns* are a set of generic approaches aiming to avoid known problems in software engineering [9].

[6] Due to space limitation, only part of the whole documentation is going to be presented in this paper. The focus will be on presenting several features available in the UMP-ST plug-in.

ii. *Evidence*: Member and responsible person of an enterprise live at the same address.

Figure 5 presents how this goal and its corresponding queries and evidence would be displayed in the UMP-ST plug-in. Note that both query and evidence are considered hypothesis in our tool. The idea is to generalize, since an evidence for a query could be another query. Therefore, we decided to call them both hypothesis.

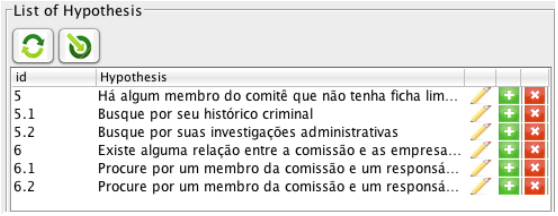


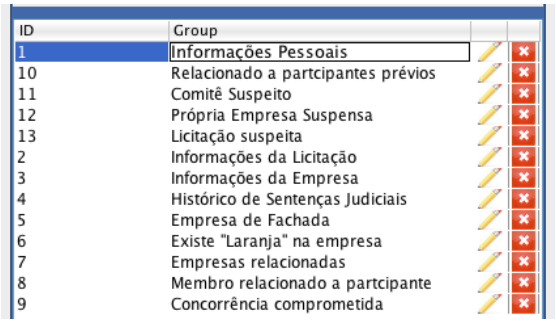
Fig. 5. Panel for displaying the hypothesis (queries and evidence) for a goal.

The next step in the POMC model is to define the entities, attributes, and relationships by looking on the set of goals/queries/evidence defined in the previous step. For instance, from the evidence that says "responsible person of an enterprise" we need to define the entities Person (Pessoa) and Enterprise (Empresa). Figure 4 presets the entity Enterprise (Empresa) with its attributes, goals and hypothesis defined as backtraking elements, as well as traceability panel with its forward-tracking elements (attributes, rules, relationships, groups, etc).

Once the entities, its attributes, and relationships are defined, we are able to define the rules for our PO. The panel for editing rules are really similar to the panel for editing entities. The difference is that we can define what type of rule it is (deterministic or stochastic). Moreover, the backtraking panel allows the user to add elements from the previous step in the POMC cycle, i.e., entities, attributes, and relationships, as well as elements in the current step, i.e., other rules. Thus, the forward-tracking panel only allows elements from the current and future steps in the process, i.e., other rules and groups.

Finally, once the rules are defined, the user can go to the final step of the Analysis & Design discipline, which is to define the groups, which will facilitate the implementation of the PO. The panel for creating groups is similar to the panel for editing rules. The difference is that the forward-tracking panel allows only other groups.

Figure 6 presents a list of groups created. Note that there is pretty much a one-to-one correspondence to the Multi-Entity Bayesian Networks Fragments (MFrags) created in [4] (see Figure 7). For instance, the Personal Information (Informações Pessoais) group is implemented as the Personal Information MFrag, the Enterprise Information (Informações da Empresa) group is implemented as the Enterprise Information MFrag, etc.



ID	Group
1	Informações Pessoais
10	Relacionado a partcipantes prévios
11	Comitê Suspeito
12	Própria Empresa Suspensa
13	Licitação suspeita
2	Informações da Licitação
3	Informações da Empresa
4	Histórico de Sentenças Judiciais
5	Empresa de Fachada
6	Existe "Laranja" na empresa
7	Empresas relacionadas
8	Membro relacionado a partcipante
9	Concorrência comprometida

Fig. 6. Panel displaying some groups.

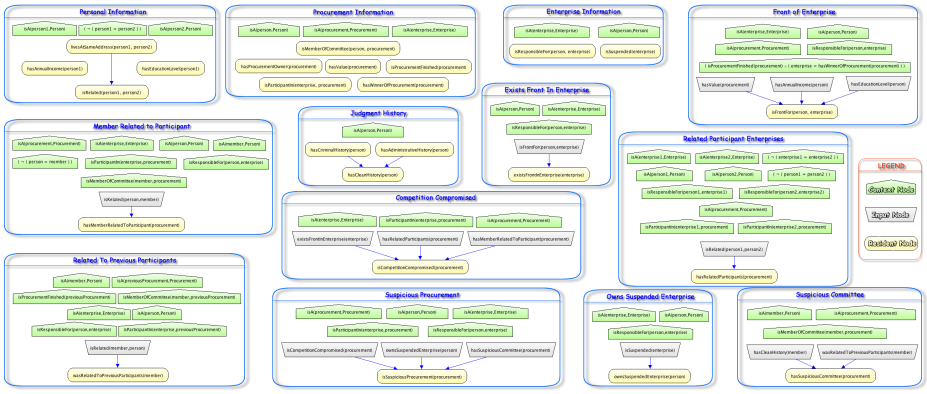


Fig. 7. Implementation of the PO in UnBBayes-MEBN.

This one-to-one mapping and the traceability feature help users deal with change and evolution of the PO. The traceability panel present when editing a goal shows all elements associated with the realization of that goal. Therefore, if a user needs to change a specific goal he/she knows where it is going to impact, all the way to the implementation. Without the UMP-ST plug-in this would be infeasible.

6 Conclusion

This paper presented the UMP-ST plug-in. A GUI tool for designing, maintaining, and evolving POs. To the best of our knowledge, this is not only the first implementation in the world of the UMP-ST process, but also the first tool to support the design of POs.

The UMP-ST plug-in provides a step by step guidance in designing POs, which allows the user to overcome the complexity in creating POs. Moreover, the plug-in also provides a centralized tool for documenting POs, whereas before

the documentation was spread in different documents (word documents with requirements, UML diagrams with entities, attributes, and relations, etc).

Finally, the difficulty in maintaining and evolving existing POs is addressed mainly by the traceability feature. The implementation of both forward-tracking and back-tracking provide a constant attention to where and what your changes might impact, which facilitates maintainability and evolution of existing POs. Although this traceability can be achieved by a simple implementation of RTM in tools like spreadsheets, as the PO becomes larger this manual traceability becomes infeasible and error prone.

The UMP-ST plug-in is still in beta phase. Some of the features that should be included in the future are: exporting all documentation to a single PDF of HTML file; and generating MFrags automatically based on the groups defined in the last step of the Analysis & Design discipline, in order to facilitate the creation of a MEBN model (*i.e.*, PR-OWL PO) during the Implementation discipline.

Acknowledgments. The authors gratefully acknowledge full support from the Brazilian Office of the Comptroller General (CGU) for the research reported in this paper.

References

1. Dean Allemang and James A. Hendler. *Semantic Web for the Working Ontologist*. Morgan Kaufmann, 2008.
2. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M Stal. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. John Wiley & Sons, Chichester, England, 1996.
3. Rommel N. Carvalho, Kathryn B. Laskey, and Paulo C. G. Costa. PR-OWL 2.0 bridging the gap to OWL semantics. In Fernando Bobillo, Paulo C. G. Costa, Claudia dAmato, Nicola Fanizzi, Kathryn B. Laskey, Kenneth J. Laskey, Thomas Lukasiewicz, Matthias Nickles, and Michael Pool, editors, *Uncertainty Reasoning for the Semantic Web II*, number 7123 in Lecture Notes in Computer Science, pages 1–18. Springer Berlin Heidelberg, January 2013.
4. Rommel Novaes Carvalho. *Probabilistic Ontology: Representation and Modeling Methodology*. PhD, George Mason University, Fairfax, VA, USA, Forthcoming.
5. Paulo C. G Costa. *Bayesian Semantics for the Semantic Web*. PhD, George Mason University, Fairfax, VA, USA, July 2005.
6. Paulo C. G Costa, Kathryn B Laskey, and Kenneth J Laskey. PR-OWL: a bayesian framework for the semantic web. In *Proceedings of the First Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2005)*, Galway, Ireland, November 2005.
7. Paulo Cesar Costa, Kathryn B. Laskey, and Kenneth J. Laskey. PR-OWL: a bayesian ontology language for the semantic web. In *Uncertainty Reasoning for the Semantic Web I: ISWC International Workshops, URSW 2005-2007, Revised Selected and Invited Papers*, pages 88–107. Springer-Verlag, 2008.
8. Zhongli Ding, Yun Peng, and Rong Pan. BayesOWL: uncertainty modeling in semantic web ontologies. In *Soft Computing in Ontologies and Semantic Web*, volume 204, pages 3–29. Springer Berlin / Heidelberg, 2006. 10.1007/978-3-540-33473-6_1.

9. E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, USA, 1994.
10. Asuncion Gomez-Perez, Oscar Corcho, and Mariano Fernandez-Lopez. *Ontological Engineering: with Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web, First Edition*. Springer, July 2004.
11. O.C.Z. Gotel and C.W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering, 1994*, pages 94–101, 1994.
12. Kevin B. Korb and Ann E. Nicholson. *Bayesian Artificial Intelligence*. Chapman & Hall/CRC, 1 edition, September 2003.
13. Marcelo Ladeira, Danilo da Silva, Mrio Vieira, Michael Onishi, Rommel Novaes Carvalho, and Wagner da Silva. Platform independent and open tool for probabilistic networks. In *Proceedings of the IV Artificial Intelligence National Meeting (ENIA 2003) on the XXIII Congress of the Brazilian Computer Society (SBC 2003)*, Unicamp, Campinas, Brazil, August 2003.
14. Kathryn Blackmond Laskey and Suzanne M. Mahoney. Network engineering for agile belief network models. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):487–498, 2000.
15. Thomas Lukasiewicz. Expressive probabilistic description logics. *Artificial Intelligence*, 172(6-7):852–883, April 2008.
16. Shou Matsumoto, Rommel Novaes Carvalho, Marcelo Ladeira, Paulo Cesar G. da Costa, Laecio Lima Santos, Danilo Silva, Michael Onishi, Emerson Machado, and Ke Cai. UnBBayes: a java framework for probabilistic models in AI. In *Java in Academia and Research*. iConcept Press, 2011.
17. Walker W. Royce. Managing the development of large software systems: Concepts and techniques. *Proceedings of IEEE WESTCON*, pages 1–9, 1970. Reprinted in Proceedings of the Ninth International Conference on Software Engineering, March 1987, pp. 328–338.
18. Ian Sommerville. *Software Engineering*. Addison Wesley, 9 edition, March 2010.
19. Karl E. Wiegers. *Software Requirements*. Microsoft Press, 2nd ed. edition, February 2003.
20. Yi Yang and Jacques Calmet. OntoBayes: an Ontology-Driven uncertainty model. In *Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-1 (CIMCA-IAWTIC'06) - Volume 01*, pages 457–463. IEEE Computer Society, 2005.