UMP-ST Plug-in: Documenting, Maintaining and Evolving Probabilistic Ontologies Using UnBBayes Framework

Rommel N. Carvalho[1], Laécio L. dos Santos[2(✉)], Marcelo Ladeira[2],
Henrique A. da Rocha[1], and Gilson L. Mendes[1]

[1] Department of Research and Strategic Information (DIE),
Brazilian Office of the Comptroller General (CGU), 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 (CIC), University of Brasília (UnB),
Campus Universitário Darcy Ribeiro, Brasília, Distrito Federal, Brazil
laecio@gmail.com, mladeira@unb.br
http://www.cic.unb.br

Abstract. Several approaches have been proposed for dealing with uncertainty in the Semantic Web (SW). Although probabilistic ontologies (PO) is one of the most promising approach to model uncertainty in ontologies, no support has been offered to ontological engineers on how to create this more complex type of ontologies. This task has proven to be extremely difficult and hard, which motivated the creation of the Uncertainty Modeling Process for Semantic Technologies (UMP-ST), a process that guides users in modeling POs. This paper presents the UMP-ST plug-in, a tool that implements this process and shows how the plug-in, implemented in UnBBayes Framework, overcomes the main problems on modeling probabilistic ontologies: the complexity in creating; the difficulty in maintaining and evolving; and the lack of a centralized tool for documenting these ontologies. The probabilistic ontology for Procurement Fraud Detection and Prevention in Brazil is used to show how the UMP-ST plug-in overcomes these problems. This probabilistic ontology is a proof-of-concept use case created as part of a research project at the Brazilian Office of the Comptroller General (CGU). (A short version of this paper was presented on the URSW 2013 [3]).

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 was motivated by the need for representation and inference in domains with uncertainty, since

OWL, the standard Web Ontology Language, supports only deterministic ontologies. This has given birth to several new languages like: PR-OWL [8–10], PR-OWL 2 [4,5], OntoBayes [26], BayesOWL [11], and probabilistic extensions of SHIF(**D**) and SHOIN(**D**) [20].

However, the increase of expressive power that 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 [1,14] and knowledge engineering for Bayesian networks [16,19], the literature contains little guidance on how to model a probabilistic ontology.

To fill the gap, Carvalho [5] proposed the Uncertainty Modeling Process for Semantic Technologies (UMP-ST), a methodology based on the Unified Process, which describes the main tasks involved in creating probabilistic ontologies incrementally and iteratively.

Nevertheless, the UMP-ST is only a guideline on things you should think about and things you should do, but it does not provide a tool for doing so. In this paper we present the UMP-ST plug-in for UnBBayes. This plug-in has the objective of dealing with three main problems: the complexity in creating probabilistic ontologies; the difficulty in maintaining and evolving existing probabilistic ontologies; and 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 the Probabilistic Web Ontology Language (PR-OWL) and the Multi-Entity Bayesian Network (MEBN), semantic technologies that motivated the creation of the UMP-ST. Section 4 presents UnBBayes and its plug-in framework. Then, Sect. 5 describes UMP-ST plug-in, which is the main contribution of this paper. Section 6 illustrates how this tool can be used to create a probabilistic ontology for procurement fraud detection and prevention. Finally, Sect. 7 presents some concluding remarks.

2   UMP-ST

The UMP-ST was proposed by Carvalho [5] as a methodology to build probabilistic ontologies. The UMP-ST is based on the Unified Process (UP), a framework that describes the activities that a team performs to transform a set of requirements into a software system [23]. Like the UP, the UMP-ST uses an iterative and incremental approach, building the ontology through several deliveries, each adding new requirements to the previous ones.

The UMP-ST divides the construction of a PO in four phases: Inception, where goals are defined; Elaboration, where the ontology will be modeled; Construction, where the ontology will be implemented; and Transition, where a new version of the ontology will be available. Inside each phase, four major disciplines guide the modeler: Requirements, Analysis & Design, Implementation and Test.

Figure 1 depicts the intensity of each discipline during the UMP-ST, which 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.
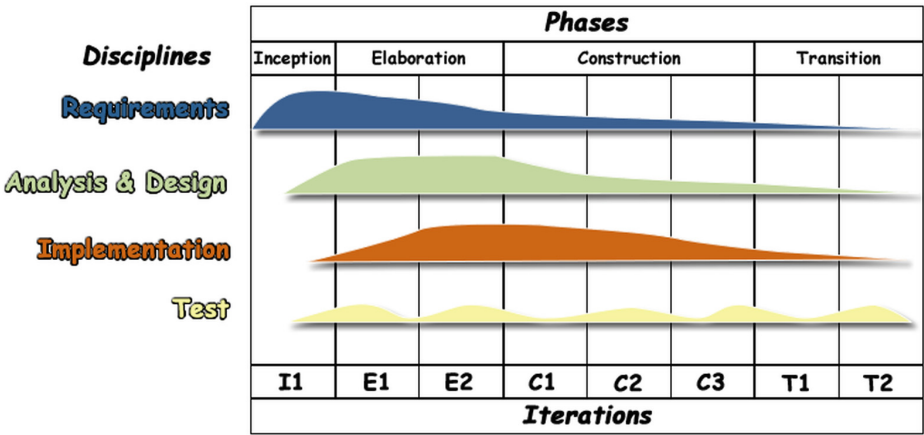

Fig. 1. Uncertainty Modeling Process for Semantic Technologies (UMP-ST).

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 [22], the POMC 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.

The **Requirements** discipline 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. This definition is independent of the language used to implement the model.
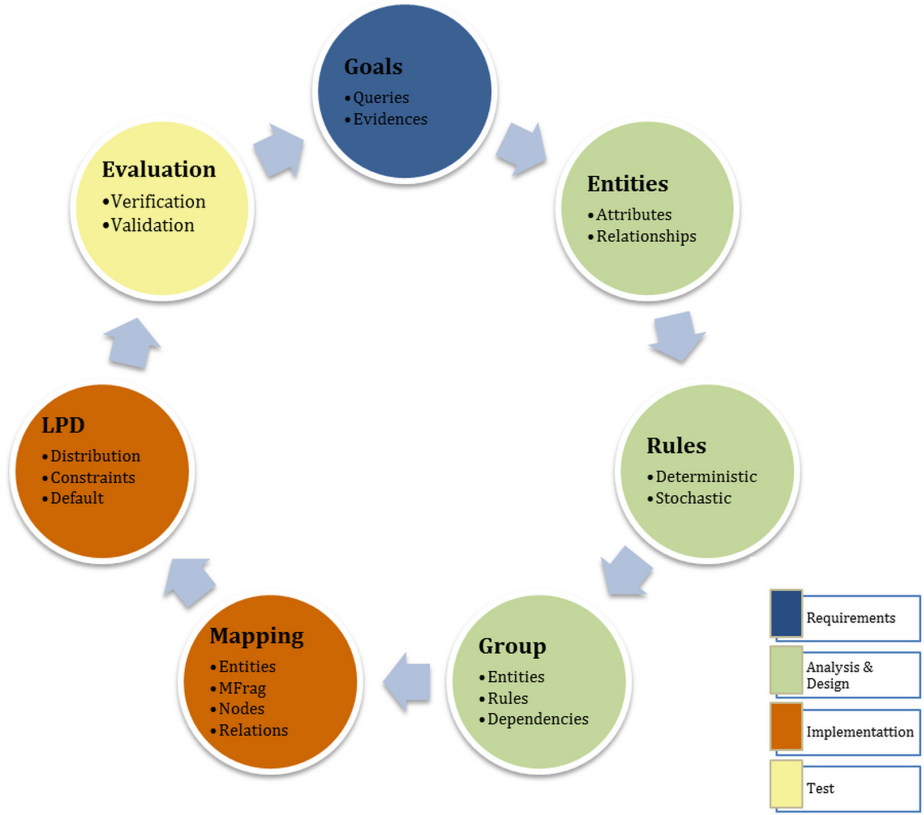
Fig. 2. Probabilistic Ontology Modeling Cycle (POMC)

The **Implementation** discipline maps our design to a specific language that allows uncertainty in semantic technologies (ST). In Fig. 2, the mapping is to PR-OWL where the elements defined during Analysis & Design will be mapping to constructions like entities, random variables, arcs and MFrags. In this discipline, the Local Probability Distribution (LPD) will also be defined. When mapping to other technologies (e.g., OntoBayes), it will be necessary to use different constructions (e.g., L-Nodes and arcs).

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. 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 [15] 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, allowing a fast visualization of the dependencies between the requirements. In the UMP-ST 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) [24,25]. 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   PR-OWL and MEBN

OWL, the standard language for creating ontologies in the Semantic Web, lacks a proper support for uncertainty representation. The great quantity of domains involving uncertainty has made urgent the creation of a language able to represent them. In this context, Paulo Costa, created the PR-OWL (Probabilistic OWL) language in 2005, extending OWL with statements that allow the creation of probabilistic ontologies [8].

PR-OWL uses the MEBN formalism to represent uncertainty in standard OWL ontologies. This is done by modeling the domain into a MEBN Theory (MTheory), a set of MEBN Fragments (MFrags) composed of nodes that represent the random variables of the model. MEBN provides a probabilistic inference based on first-order logic and Bayesian networks[1]. Figure 3 shows the main elements of PR-OWL.

MEBN is a language for representing a probabilistic knowledge based on Bayesian networks and First-Order Logic (FOL) [18]. MEBN increases the power of Bayesian networks to add the expressive power of FOL. Moreover, it extends FOL by adding a way to specify probabilistic distributions. MEBN solves the main limitation of Bayesian networks: the inability to represent situations where the number of random variables involved are unknown in advance. This limitation makes it impossible to use Bayesian networks for domains that involve recursion.

MEBN represents the domain knowledge with a MEBN Theory (MTheory). A MTheory is composed by a set of MFrags, each of which representing probability information about a group of related random variables [18]. This set of

---

[1] PR-OWL requires a MEBN inference engine to process the additional syntax.
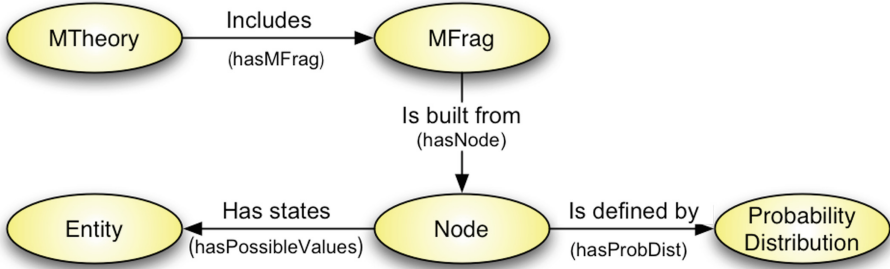
Fig. 3. PR-OWL main elements (reproduced with permission from Costa [8])

MFrags satisfies consistency constraints that guarantee the existence of a unique joint distribution over its random variables.

An MFrag is composed of resident nodes, input nodes and context nodes. Resident nodes and input nodes represent properties and relationships of entities, and have arguments (ordinary variables) that will be filled with entities during the instantiation of the model. A resident node has its LPD defined into the MFrag, while an input node is a reference to a resident node in a different MFrag. Oriented edges represent dependencies between the nodes. Context nodes define the constraints that should be observed for the probabilistic relations defined in its MFrag to be valid.

An MTheory works like a template, where MFrags will be instantiated from entities, relationships and existing evidence (findings). This instantiation will result in a Situation-Specific Bayesian Network (SSBN), where nodes of MFrags become standard Bayesian network nodes. After instantiating the SSBN, a Bayesian network inference algorithm can be used to calculate the distribution of a node of interest.

In 2011, Carvalho proposed PR-OWL 2, extending PR-OWL to provide a formal mapping between OWL concepts and PR-OWL random variables and to make the types already present in OWL compatible with PR-OWL [5]. PR-OWL 2 made it easier to construct hybrid ontologies containing probabilistic and deterministic statements.

4   UnBBayes Plug-in Architecture

UnBBayes is an open-source Java^TM 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 provides a high degree of scalability [17,21]. Figure 4 shows a screenshot of the tool with several plug-ins opened as different internal windows.
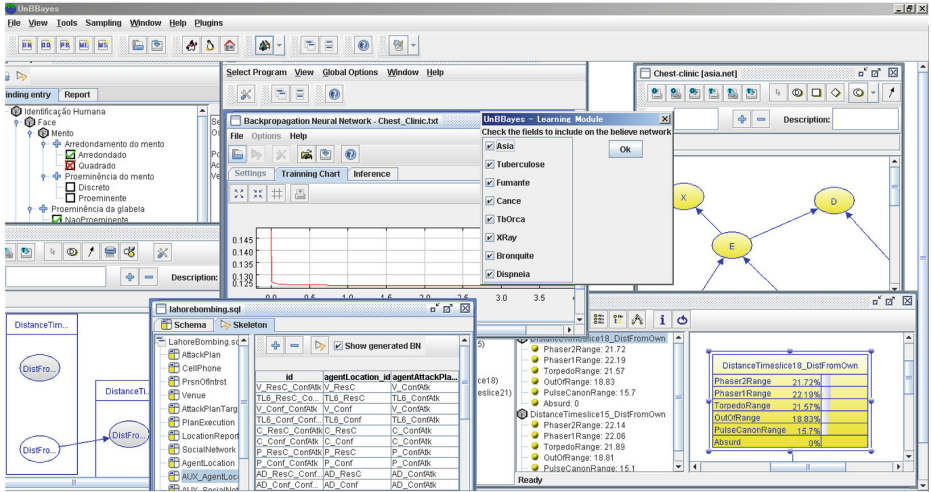
Fig. 4. Screenshot of the framework UnBBayes with several plug-ins.

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**[2] (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[3] (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

---

[2] 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.

[3] http://jpf.sourceforge.net/

(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 5 illustrates the main classes of a module plug-in. `UnBBayesModule` is the most important class of a module plug-in 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 [21].

UnBBayes provides plug-ins for various formalisms based on Bayesian Networks, including Influence Diagram (ID), Multiply-Sectioned Bayesian Network (MSBN), Hybrid Bayesian Network (HBN), Object-Oriented Bayesian Network (OOBN), Probabilistic Relational Model (PRM), and Multi-Entity Bayesian Network (MEBN).

UnBBayes was the first tool to implement MEBN. It has a GUI for creating the model graphically, a knowledge base for representation and reasoning in FOL, a language for specifying the LPD's, and an algorithm for generating the SSBN from a set of queries and findings [6]. In this first version, UnBBayes uses PR-OWL format to persist the model, and PowerLoom [7] as the Knowledge Base system. Recently, a plug-in for PR-OWL 2 was implemented, integrating Protégé [13], a free, open-source ontology editor, to UnBBayes, allowing the user to build both the deterministic and the probabilistic part of a ontology using the UnBBayes.

5   UMP-ST Plug-in

The UMP-ST tool was implemented by the Artificial Intelligence Group of the University of Brasilia as a plug-in for UnBBayes. As seen in Sect. 2, the UMP-ST process consists of four major disciplines: Requirements, Analysis & Design, Implementation, and Test. Nevertheless, the UMP-ST plug-in focuses only on the Requirements and Analysis & Design disciplines, since they are the only language independent disciplines. As seen in Sect. 4, UnBBayes has plug-ins for building probabilistic ontologies in PR-OWL and PR-OWL 2, which can be used in the Implementation and Test disciplines.

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
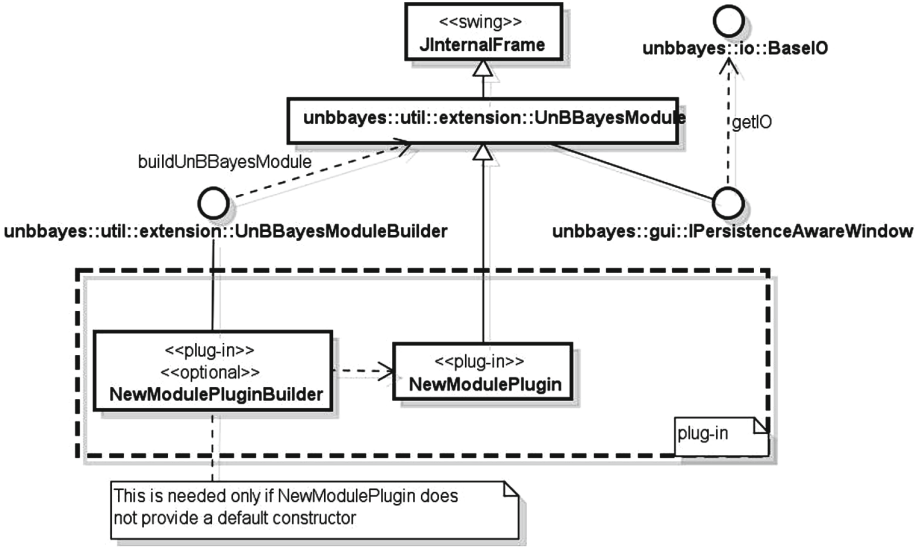3. The lack of a centralized tool for documenting probabilistic ontologies.

Fig. 5. Class diagram of classes that must be extended to create a module plug-in.

The UMP-ST plug-in is almost like a wizard tool that guides the user in each and every step of the Requirements and Analysis & Design disciplines. The user begins defining 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. Then the user is allowed to move to the next phase of the process which is defining the entities, with their attributes and relationships. After this, he should define rules, both probabilistic and deterministic. Finally he should define the groups related to the defined goals, queries, and evidence (see Fig. 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. This documentation is essential in every project, since it allows other users to quickly capture the solution and the reasons that led it to be built that way. The plug-in allows the user to enter comments for each editable element and storing information about the author and creation date.

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 backtracking panel shows every element from previous steps of the process associated with the element being edited. On the other hand, the forwardtracking 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 changes might impact, which facilitates maintainability and evolution of existing POs.

The Fig. 6 shows the initial pane for editing entities in the plug-in. As shown, the GUI opens inside the UnBBayes window as a new frame. There are four tabs available: to edit the Requirements, the user utilizes the tab Goals; to edit the Analysis & Design, the user utilizes the tabs Entities, Rules and Groups. Selecting a tab initially opens a panel listing all elements of that type (in Fig. 6 all entities are listed, since the Entities tab was selected). The user can then choose to edit or to delete an existing element, or to add new elements. In the tab Entities there is also a button that opens the panel for editing relationships.

Figure 7 presents the panel for editing entities with some of the main features of the UMP-ST plug-in. Note the backtracking panel that in this case shows which goals and hypothesis originated the creation of the entity. The fowardtracking panel lists all elements created from this entity (attributes, relationships, rules and groups).

The UMP-ST tool was implemented with the Java programming language, using the Swing API for developing the user interface. The plug-in is distributed under GPL license, and it is available on sourceforge[4]. Since it is a module plug-in in UnBBayes, the `UmpstModule` class extends the `UnBBayesModule` class. The UMP-ST plug-in is mostly structured in a Model-View-Controller (MVC[5]) design pattern[6], 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. In this current beta version of the plug-in, the project is saved using the serialization method, in which an image of Java objects is saved into a file, allowing it to be reassembled during the load process.

---

[4] http://sourceforge.net/projects/unbbayes/

[5] An 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].

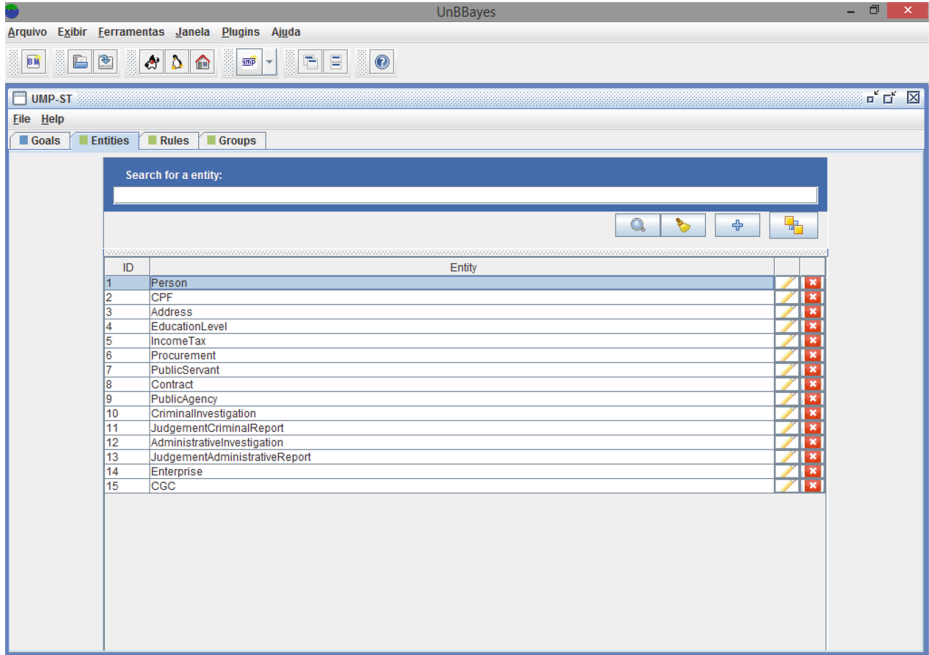[6] Design patterns are a set of generic approaches aiming to avoid known problems in software engineering [12].

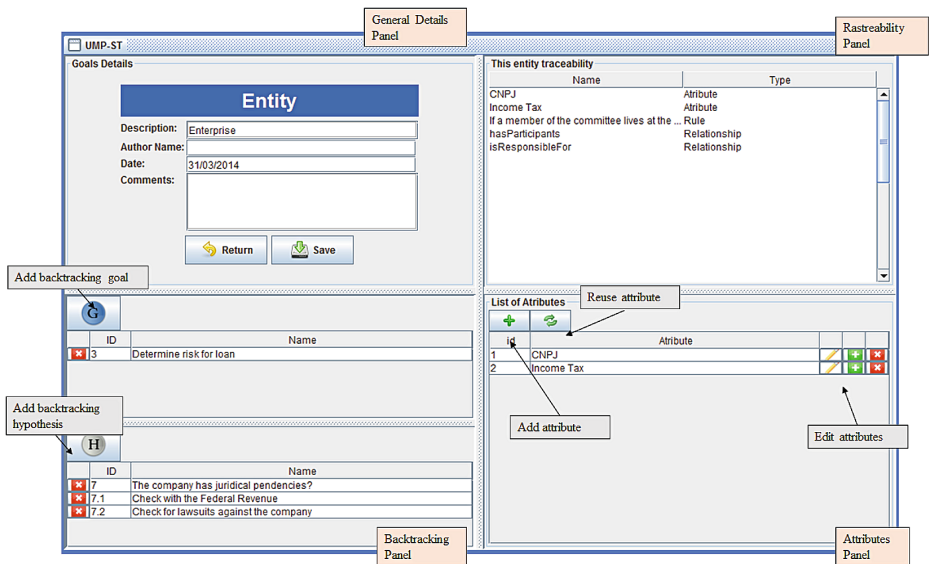**Fig. 6.** Initial Panel of the Tab for editing Entities

Fig. 7. Panel for editing an entity with a few features highlighted.

6   Use Case

The contract bidding process is one of the main process that can be used for corruption actions in Brazil. Although there are laws that try to ensure competitiveness and a fair process, perpetrators find ways to turn the process to their advantage while appearing to be legitimate. The probabilistic ontology described in this section was created by Carvalho [5] based on information about different types of fraud encountered by the Brazilian Office of the Comptroller General (CGU), didactically structured by experts. The purpose of this ontology is to structure knowledge in a way that an automated system can reason in a similar way as a specialist.

This section presents how the UMP-ST plug-in can be used to build the probabilistic ontology for Procurement Fraud Detection and Prevention in Brazil, an use case presented by Carvalho [5]. Although Carvalho [5] 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[7].

As explained in Sect. 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.

Public procurements are very complex and involve large sums of money. Therefore, members forming the committee must not only be prepared, but must also have a clean history, in order to maximize the morality, one of the ethical principles that federal, state, municipal, and district governments must adopt.

One of the principles established for the contract bidding is equality among bidders. This principle prohibits the agent proxy to discriminate between potential suppliers. If an agent of the procurement has some form or relationship with a bidder, he can provide information or set new requirements in order to favor that bidder, and therefore he should not participate in this bid committee.

One of the goals of the use case is to identify whether it is needed to change the committee of a procurement, because it has a member with a dirty history or because it has a member related with any of the competitors.

One of the goals presented in [5] 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?

---

[7] 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.

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

Figure 8 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, and an evidence can be supported by sub-evidences. Therefore, we decided to call them both hypothesis, which, defined hierarchically will assemble an implicit specification tree.
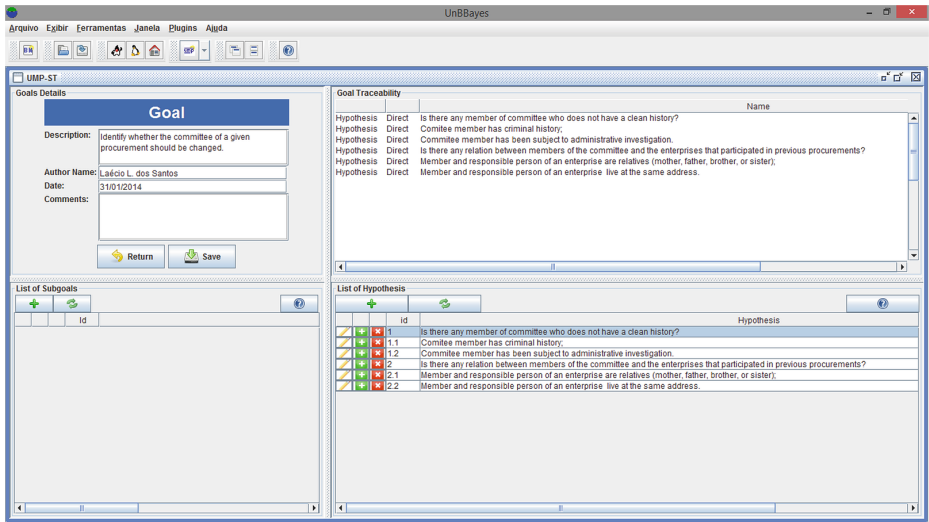
Fig. 8. Panel for displaying the hypothesis (queries and evidence) for a goal.

Note that for the definition of the requirements, the user does not need have knowledge about probabilistic ontologies, or even about semantic technologies. Thus, these can be defined by the domain expert himself. In fact, the next discipline, Analysis & Design can be done by someone with minimal knowledge about ontologies, leaving only the implementation and testing disciplines for the expert in the chosen technology.

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 and Enterprise, besides the relationship isResponsibleFor.

Figure 7 presents the entity Enterprise 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.). Relationships are edited under its own panel, where the user must name it and set which entities are involved.

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 forwardtracking panel only allows elements from the current and future steps in the process, i.e., other rules and groups.

The rules presented in [5] for the goal previously described are:

1. If a member of the committee has a relative (mother, father, brother, or sister) responsible for a bidder in the procurement, then it is more likely that a relationship exists between the committee and the enterprises, which inhibits competition.
2. If a member of the committee lives at the same address as a person responsible for a bidder in the procurement, then it is more likely that a relationship exists between the committee and the enterprises, which lowers competition.
3. If a member of the committee has been convicted of a crime or has been penalized administratively, then he/she does not have a clean history. If he/she was recently investigated, then it is likely that he/she does not have a clean history.
4. If the relation defined in 1 and 2 is found in previous procurements, then it is more likely that there will be a relation between this committee and future bidders.
5. If 3 or 4, then it is more likely that the committee needs to be changed.

As it can be seen, rules 4 and 5 illustrate how a rule can be defined from others rules. Thus, we would add rules 1 and 2 in tracking list of rule 4 and rules 3 and 4 in the tracking list of rule 5.

Figure 9 shows the panel for editing rule 2. The backtracking panel shows that this rule involves the entities Person, Address, Enterprise and Procurement, which in turn are used in the relationships livesAt, isResponsibleFor, and hasPartipants. This information will be useful on the implementation of this rule.

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 forwardtracking panel shows only other groups. Figure 10 presents a list of groups created.

Fig. 9. Rules panel.

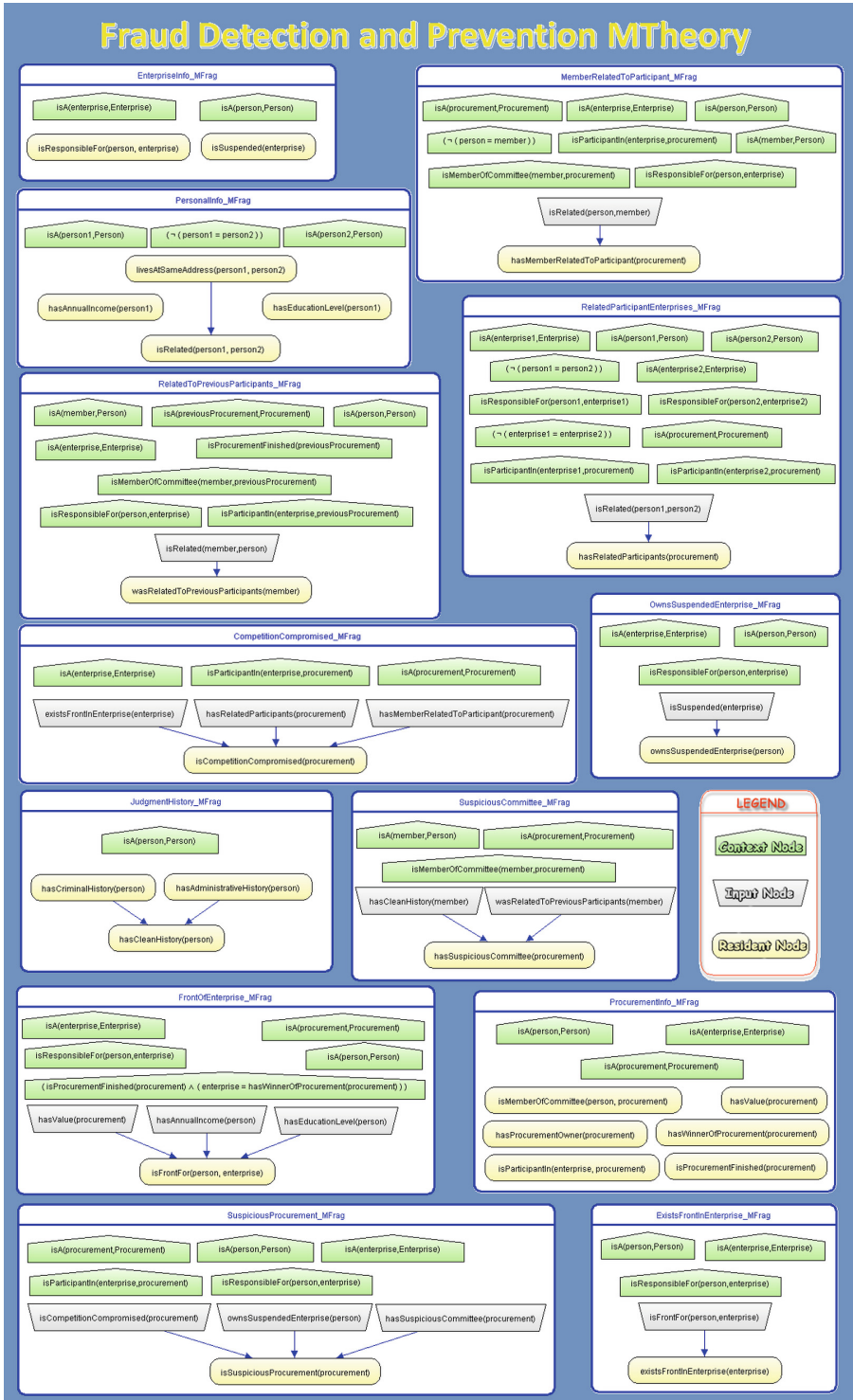**Fig. 10.** Panel displaying some groups.

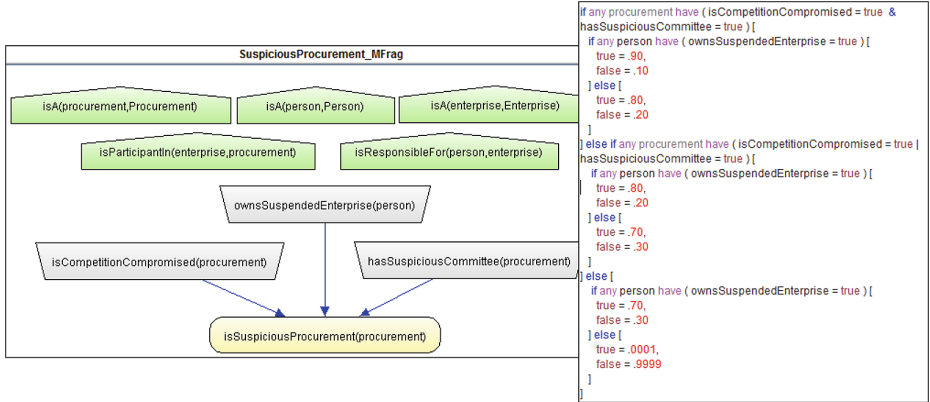**Fig. 11.** Implementation of the PO in UnBBayes-MEBN.
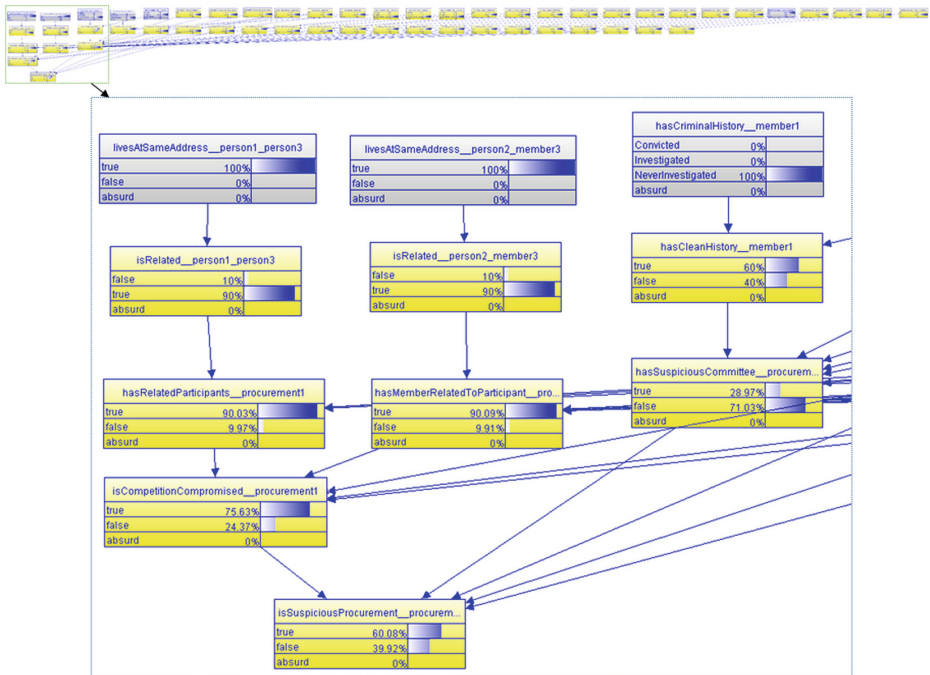
Fig. 12. LPD of the node `isRelated`.

Fig. 13. SSBN for the query `IsSuspiciousProcurement(procurement1)`.

Figure 11 shows the MTheory created in the implementation of this ontology in MEBN[8], suggested in [5]. Note that there is pretty much a one-to-one correspondence between the groups defined in UMP-ST and MFrags created in

---

[8] Avaiable in https://sourceforge.net/projects/unbbayes/files/examples/.

MTheory. For instance, the Personal Information group is implemented as the Personal Information MFrag, the Enterprise Information group is implemented as the Enterprise Information MFrag, etc.

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.

When implementing the model using PR-OWL, the entities defined during Analysis & Design will be simply created as PR-OWL entities, while their attributes and relationships will be mapped to random variables. The relationship isResponsibleFor, for example, will become the isResponsibleFor(person, enterprise) resident node into the MFrag Enterprise Information. The rules define the dependencies between the nodes and also the context nodes that will set the constraints that must be followed in order to be able to instantiate its corresponding MFrag.

The LPDs are also defined in the Implementation discipline. Although LPDs are not foreseen in the Analysis & Design discipline, it is recommended that experts add comments to each rule in order to guide how these distributions should be defined during the Implementation discipline.

Figure 12 shows the LPD for the resident node `isSuspiciousProcurement` implemented using the LPD definition language sintax used in UnBBayes. The probabilities are calculated based on the informations about the parent nodes. The assign values of this LPD are notional only, since in this use case no real data or statistics was used. When generating the SSBN, this LPD will be instanciated based on the entities present in the model, and then will became the CPT of the bayesian nodes. Figure 13 present part of a SSBN generated to answer the query `isSuspiciousProcurement(procurement1)` in a situation with three enterprises competing for the `procurement1`, with `person1`, `person2`, and `person3` responsibles for each one, and a comission formed by three members. `Person1` and `person2` live at same address. The same occurs with `member2` and `person2`. `Member1` have administrative history. For this situation, the probability of the procurement be suspicious is high (60 %).

7   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 the first tool in the world 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 forwardtracking and backtracking provide a constant attention to where and what 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: saving the project as an xml file; exporting all documentation to a single PDF of HTML file; allowing the user to edit the rules in a more visual way; and generating MTheories automatically based on the entities, attributes, relationships and groups defined in the Analysis & Design discipline, in order to facilitate the creation of a MEBN model (i.e., PR-OWL PO) during the Implementation discipline.

References

1. Allemang, D., Hendler, J.A.: Semantic Web for the Working Ontologist. Morgan Kaufmann, San Francisco (2008)
2. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture. A System of Patterns, vol. 1. Wiley, Chichester (1996)
3. Carvalho, R.N., Ladeira, M., de Souza, R.M., Matsumoto, S., Da Rocha, H.A., Mendes, G.L.: UMP-ST plug-in: a Tool for documenting, maintaining, and evolving probabilistic ontologies. In: Bobillo, F., Carvalho, R.N., da Costa, P.C.G., d'Amato, C., Fanizzi, N., Laskey, K.B., Laskey, K.J., Lukasiewicz, T., Martin, T., Nickles, M., Pool, M. (eds.) Proceedings of the 9th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2013). CEUR Workshop Proceedings, vol. 1073, pp. 15–26 (2013). CEUR-WS.org
4. Carvalho, R.N., Laskey, K.B., Costa, P.C.G.: PR-OWL 2.0 – bridging the gap to OWL semantics. In: Bobillo, F., Costa, P.C.G., d'Amato, C., Fanizzi, N., Laskey, K.B., Laskey, K.J., Lukasiewicz, T., Nickles, M., Pool, M. (eds.) URSW 2008-2010/UniDL 2010. LNCS, vol. 7123, pp. 1–18. Springer, Heidelberg (2013)
5. Carvalho, R.N.: Probabilistic ontology: representation and modeling methodology. Ph.D., George Mason University, Fairfax, VA, USA (2011)
6. Carvalho, R.N., Santos, L.L., Ladeira, M., Costa, P.C.G.: A GUI tool for plausible reasoning in the semantic web using MEBN. In: Proceedings of the Seventh International Conference on Intelligent Systems Design and Applications, ISDA '07, Los Alamitos, CA, USA, pp. 381–386. IEEE Computer Society, October 2007
7. Chalupsky, H., MacGregor, R.M., Russ, T.: Powerloom manual (2010)
8. Costa, P.C.G.: Bayesian semantics for the semantic Web. Ph.D., George Mason University, Fairfax, VA, USA (2005)
9. Costa, P.C.G., Laskey, K.B., Laskey, K.J.: 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

10. da Costa, P.C.G., Laskey, K.B., Laskey, K.J.: PR-OWL: a Bayesian ontology language for the semantic Web. In: da Costa, P.C.G., d'Amato, C., Fanizzi, N., Laskey, K.B., Laskey, K.J., Lukasiewicz, T., Nickles, M., Pool, M. (eds.) URSW 2005 - 2007. LNCS (LNAI), vol. 5327, pp. 88–107. Springer, Heidelberg (2008)

11. Ding, Z., Peng, Y., Pan, R.: BayesOWL: uncertainty modeling in semantic web ontologies. In: Ma, Z. (ed.) Soft Computing in Ontologies and Semantic Web. Studies in Fuzziness and Soft Computing, vol. 204, pp. 3–29. Springer, Heidelberg (2006). doi:10.1007/978-3-540-33473-6_1

12. Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, New York (1994)

13. Gennari, J.H., Musen, M.A., Fergerson, R.W., Grosso, W.E., Crubzy, M., Eriksson, H., Noy, N.F., Tu, S.W.: The evolution of protégé: an environment for knowledge-based systems development. Int. J. Hum.-Comput. Stud. **58**(1), 89–123 (2003)

14. Gomez-Perez, A., Corcho, O., Fernandez-Lopez, M.: Ontological Engineering: With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web, 1st edn. Springer, Heidelberg (2004)

15. Gotel, O.C.Z., Finkelstein, C.W.: An analysis of the requirements traceability problem. In: 1994 Proceedings of the First International Conference on Requirements Engineering, pp. 94–101 (1994)

16. Korb, K.B., Nicholson, A.E.: Bayesian Artificial Intelligence. Chapman & Hall/CRC, Boca Raton (2003)

17. Ladeira, M., da Silva, D., Vieira, M., Onishi, M., Carvalho, R.N., da Silva, W.: 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

18. Laskey, K.B.: MEBN: a language for first-order Bayesian knowledge bases. Artif. Intell. **172**(2–3), 140–178 (2008)

19. Mahoney, M.: Network engineering for agile belief network models. IEEE Trans. Knowl. Data Eng. **12**(4), 487–498 (2000)

20. Lukasiewicz, T.: Expressive probabilistic description logics. Artif. Intell. **172**(6–7), 852–883 (2008)

21. Matsumoto, S., Carvalho, R.N., Ladeira, M., da Costa, P.C.G., Santos, L.L., Silva, D., Onishi, M., Machado, E.: UnBBayes: a java framework for probabilistic models in AI. In: Cai, K. (ed.) Java in Academia and Research. iConcept Press, Annerley (2011)

22. Royce, W.W.: Managing the development of large software systems: concepts and techniques. In: Proceedings of IEEE WESTCON, pp. 1–9 (1970). Reprinted in Proceedings of the Ninth International Conference on Software Engineering, pp. 328–338, March 1987

23. Scott, K.: The Unified Process Explained. Addison-Wesley Longman Publishing Co Inc., Boston (2002)

24. Sommerville, I.: Software Engineering, 9th edn. Addison Wesley, Boston (2010)

25. Wiegers, K.E.: Software Requirements, 2nd edn. Microsoft Press, Redmond (2003)

26. Yang, Y., Calmet, J.: 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 (CIMCA-IAWTIC'06), vol. 1, pp. 457–463. IEEE Computer Society (2005)