9th International Workshop on Uncertainty Reasoning for the Semantic Web

Proceedings

edited by	Fernando Bobillo
	Rommel Carvalho
	Paulo C. G. da Costa
	Claudia d'Amato
	Nicola Fanizzi
	Kathryn B. Laskey
	Kenneth J. Laskey
	Thomas Lukasiewicz
	Trevor Martin
	Matthias Nickles
	Michael Pool

Sydney, Australia, October 21, 2013

collocated with
the 12th International Semantic Web Conference
– ISWC 2013 –

Foreword

This volume contains the papers presented at the 9th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2013), held as a part of the 12th International Semantic Web Conference (ISWC 2013) at Sydney, Australia, October 21, 2013. It contains two technical papers and four position papers, which were selected in a rigorous reviewing process, where each paper was reviewed by at least four program committee members.

The International Semantic Web Conference is a major international forum for presenting visionary research on all aspects of the Semantic Web. The International Workshop on Uncertainty Reasoning for the Semantic Web is an exciting opportunity for collaboration and cross-fertilization between the uncertainty reasoning community and the Semantic Web community. Effective methods for reasoning under uncertainty are vital for realizing many aspects of the Semantic Web vision, but the ability of current-generation Web technology to handle uncertainty is extremely limited. Thus, there is a continuing demand for uncertainty reasoning technology among Semantic Web researchers and developers, and the URSW workshop creates a unique opening to bring together two communities with a clear commonality of interest but limited history of interaction. By capitalizing on this opportunity, URSW could spark dramatic progress toward realizing the Semantic Web vision.

We wish to thank all authors who submitted papers and all workshop participants for fruitful discussions. Special thanks also to Anne Cregan-Wolfe for her invited talk on “Knowing our Unknowns: Butterflies’ Wings, Black Swans, Buckley’s Chance and the Last Japanese Soldier.” Furthermore, we would like to thank the program committee members and external referees for their timely expertise in carefully reviewing the submissions.

October 2013

Fernando Bobillo
Rommel Carvalho
Paulo C. G. da Costa
Claudia d’Amato
Nicola Fanizzi
Kathryn B. Laskey
Kenneth J. Laskey
Thomas Lukasiewicz
Trevor Martin
Matthias Nickles
Michael Pool

Workshop Organization

Program Chairs

Fernando Bobillo (University of Zaragoza, Spain)
Rommel Carvalho (George Mason University, USA)
Paulo C. G. da Costa (George Mason University, USA)
Claudia d'Amato (University of Bari, Italy)
Nicola Fanizzi (University of Bari, Italy)
Kathryn B. Laskey (George Mason University, USA)
Kenneth J. Laskey (MITRE Corporation, USA)
Thomas Lukasiewicz (University of Oxford, UK)
Trevor Martin (University of Bristol, UK)
Matthias Nickles (National University of Ireland, Ireland)
Michael Pool (Goldman Sachs, USA)

Program Committee

Fernando Bobillo (University of Zaragoza, Spain)
Silvia Calegari (University of Milano-Bicocca, Italy)
Rommel Carvalho (George Mason University, USA)
Davide Ceolin (Vrije Universiteit Amsterdam, Netherlands)
Paulo C. G. da Costa (George Mason University, USA)
Fabio Gagliardi Cozman (Universidade de São Paulo, Brazil)
Claudia d'Amato (University of Bari, Italy)
Nicola Fanizzi (University of Bari, Italy)
Marcelo Ladeira (Universidade de Brasília, Brazil)
Kathryn B. Laskey (George Mason University, USA)
Kenneth J. Laskey (MITRE Corporation, USA)
Thomas Lukasiewicz (University of Oxford, UK)
Trevor Martin (University of Bristol, UK)
Alessandra Mileo (National University of Ireland, Ireland)
Matthias Nickles (National University of Ireland, Ireland)
Jeff Z. Pan (University of Aberdeen, UK)
Rafael Peñaloza (TU Dresden, Germany)
Michael Pool (Goldman Sachs, USA)
Livia Predoiu (University of Oxford, UK)

Table of Contents

Technical Papers

Information Integration with Provenance on the Semantic Web via Probabilistic Datalog$^{\pm}$

Thomas Lukasiewicz and Livia Predoiu

Department of Computer Science, University of Oxford, UK
`firstname.lastname@ox.cs.ac.uk`

Abstract. The recently introduced Datalog$^{\pm}$ family of tractable knowledge representation formalisms is able to represent and reason over light-weight ontologies. It extends plain Datalog by negative constraints and the possibility of rules with existential quantification and equality in rule heads, and at the same time restricts the rule syntax by the addition of so-called guards in rule bodies to gain decidability and tractability. In this paper, we investigate how a recently proposed probabilistic extension of Datalog$^{\pm}$ can be used for representing ontology mappings in typical information integration settings, such as data exchange, data integration, and peer-to-peer integration. To allow to reconstruct the history of the mappings, to detect cycles, and to enable mapping debugging, we also propose to extend it by provenance annotations.

1 Introduction

Information integration aims at querying in a uniform way information that is distributed over multiple heterogeneous sources. This is usually done via mappings between logical formalizations of data sources such as database schemas, ontology schemas, or TBoxes; see also [16, 9, 21]. It is commonly agreed that there are mainly three principles on how data or information from different sources can be integrated:

- **Data exchange:** Data structured under a source schema S (or more generally under different source schemas $S_1, \ldots, S_k$) is transformed into data structured under a different target schema T and materialized (merged and acquired) there through the mapping.
- **Data integration:** Heterogeneous data in different sources $S_1, \ldots, S_k$ is queried via a virtual global schema T, i.e., no actual exchange of data is needed.
- **Peer-to-peer data integration:** There is no global schema given. All peers $S_1, \ldots,$ S_k are autonomous and independent from each other, and each peer can hold data and be queried. The peers can be viewed as nodes in a network that are linked to other nodes by means of so-called peer-to-peer (P2P) mappings. That is, each source can also be a target for another source.

Recently, a probabilistic extension of Datalog$^{\pm}$ [11] has been introduced, which we here propose to use as a mapping language in the above information integration scenarios. Classical Datalog$^{\pm}$ [2] combines Datalog with negative constraints and tuple- and equality-generating dependencies (TGDs and EGDs, respectively) under certain restrictions to gain decidability and data tractability. In this way, it is possible to capture the *DL-Lite* family of description logics and also the description logic $\mathcal{EL}$. The probabilistic extension is based on Markov logic networks (MLNs) [19].

In this paper, we investigate how probabilistic Datalog$^{\pm}$ can be used as a mapping language for information integration and propose to add provenance information to mappings to be able to track the origin of a mapping for trust assessment and debugging. Capturing the provenance of mappings allows to resolve inconsistencies of mappings by considering the history of their creation. Furthermore, it helps to detect whether and how to perform mapping updates in case the information sources have changed or evolved. Finally, it allows to capture mapping cycles, debug mappings and to perform meta-reasoning with mappings and the knowledge bases themselves.

2 Guarded Datalog$^{\pm}$

We now describe guarded Datalog$^{\pm}$ [2], which here includes negative constraints and (separable) equality-generating dependencies (EGDs). We first describe some preliminaries on databases and queries, and then tuple-generating dependencies (TGDs) and the concept of chase. We finally recall negative constraints and (separable) EGDs, which are other important ingredients of guarded Datalog$^{\pm}$ ontologies.

2.1 Databases and Queries

For the elementary ingredients, we assume data constants, nulls, and variables as follows; they serve as arguments in atomic formulas in databases, queries, and dependencies. We assume (i) an infinite universe of *data constants* Δ (which constitute the "normal" domain of a database), (ii) an infinite set of *(labeled) nulls* Δ_N (used as "fresh" Skolem terms, which are placeholders for unknown values, and can thus be seen as variables), and (iii) an infinite set of variables $\mathcal{V}$ (used in queries and dependencies). Different constants represent different values (*unique name assumption*), while different nulls may represent the same value. We assume a lexicographic order on $\Delta \cup \Delta_N$, with every symbol in Δ_N following all symbols in Δ. We denote by $\mathbf{X}$ sequences of variables $X_1, \ldots, X_k$ with $k \geqslant 0$.

We next define atomic formulas, which occur in databases, queries, and dependencies, and which are constructed from relation names and terms, as usual. We assume a *relational schema* $\mathcal{R}$, which is a finite set of *relation names* (or *predicate symbols*, or simply *predicates*). A *position* $P[i]$ identifies the i-th argument of a predicate P. A *term* t is a data constant, null, or variable. An *atomic formula* (or *atom*) $\mathbf{a}$ has the form $P(t_1, ..., t_n)$, where P is an n-ary predicate, and $t_1, ..., t_n$ are terms. We denote by $pred(\mathbf{a})$ and $dom(\mathbf{a})$ its predicate and the set of all its arguments, respectively. The latter two notations are naturally extended to sets of atoms and conjunctions of atoms. A conjunction of atoms is often identified with the set of all its atoms.

We are now ready to define the notion of a database relative to a relational schema, as well as conjunctive and Boolean conjunctive queries to databases. A *database (instance) D* for a relational schema $\mathcal{R}$ is a (possibly infinite) set of atoms with predicates from $\mathcal{R}$ and arguments from Δ. Such D is *ground* iff it contains only atoms with arguments from Δ. A *conjunctive query (CQ)* over $\mathcal{R}$ has the form $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms with the variables $\mathbf{X}$ and $\mathbf{Y}$, and eventually constants, but without nulls. Note that $\Phi(\mathbf{X}, \mathbf{Y})$ may also contain equalities but no inequalities. A *Boolean CQ (BCQ)* over $\mathcal{R}$ is a CQ of the form $Q()$. We often write a BCQ as the set of all its atoms, having constants and variables as arguments, and omitting the quantifiers. Answers to CQs and BCQs are defined via *homomorphisms*, which are mappings $\mu\colon \Delta \cup \Delta_N \cup \mathcal{V} \to \Delta \cup \Delta_N \cup \mathcal{V}$ such that (i) $c \in \Delta$ implies $\mu(c) = c$, (ii) $c \in \Delta_N$ implies $\mu(c) \in \Delta \cup \Delta_N$, and (iii) μ is naturally extended to atoms, sets of atoms, and conjunctions of atoms. The set of all *answers* to a CQ $Q(\mathbf{X}) = \exists \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y})$ over a database D, denoted $Q(D)$, is the set of all tuples $\mathbf{t}$ over Δ for which there exists a homomorphism $\mu\colon \mathbf{X} \cup \mathbf{Y} \to \Delta \cup \Delta_N$ such that $\mu(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$ and $\mu(\mathbf{X}) = \mathbf{t}$. The *answer* to a BCQ $Q()$ over a database D is *Yes*, denoted $D \models Q$, iff $Q(D) \neq \emptyset$.

2.2 Tuple-Generating Dependencies

Tuple-generating dependencies (TGDs) describe constraints on databases in the form of generalized Datalog rules with existentially quantified conjunctions of atoms in rule heads; their syntax and semantics are as follows. Given a relational schema $\mathcal{R}$, a *tuple-generating dependency (TGD)* σ is a first-order formula of the form $\forall \mathbf{X} \forall \mathbf{Y}\, \Phi(\mathbf{X}, \mathbf{Y}) \to \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$, where $\Phi(\mathbf{X}, \mathbf{Y})$ and $\Psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$ called the *body* and the *head* of σ, denoted $\mathit{body}(\sigma)$ and $\mathit{head}(\sigma)$, respectively. A TGD is *guarded* iff it contains an atom in its body that involves all variables appearing in the body. The leftmost such atom is the *guard atom* (or *guard*) of σ. The non-guard atoms in the body of σ are the side atoms of σ. We usually omit the universal quantifiers in TGDs. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D, there exists an extension h' of h that maps the atoms of $\Psi(\mathbf{X}, \mathbf{Z})$ to atoms of D. All sets of TGDs are finite here.

Query answering under TGDs, i.e., the evaluation of CQs and BCQs on databases under a set of TGDs is defined as follows. For a database D for $\mathcal{R}$, and a set of TGDs Σ on $\mathcal{R}$, the set of *models* of D and Σ, denoted $\mathit{mods}(D, \Sigma)$, is the set of all (possibly infinite) databases B such that (i) $D \subseteq B$ (ii) every $\sigma \in \Sigma$ is satisfied in B. The set of *answers* for a CQ Q to D and Σ, denoted $\mathit{ans}(Q, D, \Sigma)$, is the set of all tuples $\mathbf{a}$ such that $\mathbf{a} \in Q(B)$ for all $B \in \mathit{mods}(D, \Sigma)$. The *answer* for a BCQ Q to D and Σ is *Yes*, denoted $D \cup \Sigma \models Q$, iff $\mathit{ans}(Q, D, \Sigma) \neq \emptyset$. We recall that query answering under TGDs is equivalent to query answering under TGDs with only single atoms in their heads. We thus often assume w.l.o.g. that every TGD has a single atom in its head.

2.3 The Chase

The *chase* was introduced to enable checking implication of dependencies [17] and later also for checking query containment [14]. It is a procedure for repairing a database relative to a set of dependencies, so that the result of the chase satisfies the dependencies. By "chase", we refer both to the chase procedure and to its output. The TGD chase works on a database through so-called TGD *chase rules* (an extended chase with also equality-generating dependencies is discussed below). The TGD chase rule comes in two flavors: *restricted* and *oblivious*, where the restricted one applies TGDs only when they are not satisfied (to repair them), while the oblivious one always applies TGDs (if they produce a new result). We focus on the oblivious one here; the *(oblivious) TGD chase rule* defined below is the building block of the chase.

TGD CHASE RULE. Consider a database D for a relational schema $\mathcal{R}$, and a TGD σ on $\mathcal{R}$ of the form $\Phi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \Psi(\mathbf{X}, \mathbf{Z})$. Then, σ is *applicable* to D if there exists a homomorphism h that maps the atoms of $\Phi(\mathbf{X}, \mathbf{Y})$ to atoms of D. Let σ be applicable to D, and h_1 be a homomorphism that extends h as follows: for each $X_i \in \mathbf{X}$, $h_1(X_i) = h(X_i)$; for each $Z_j \in \mathbf{Z}$, $h_1(Z_j) = z_j$, where z_j is a "fresh" null, i.e., $z_j \in \Delta_N$, z_j does not occur in D, and z_j lexicographically follows all other nulls already introduced. The *application of* σ on D adds to D the atom $h_1(\Psi(\mathbf{X}, \mathbf{Z}))$ if not already in D. ■

The chase algorithm for a database D and a set of TGDs Σ consists of an exhaustive application of the TGD chase rule in a breadth-first (level-saturating) fashion, which leads as result to a (possibly infinite) chase for D and Σ. Formally, the *chase of level up to* 0 of D relative to Σ, denoted $\mathit{chase}^0(D, \Sigma)$, is defined as D, assigning to every atom in D the *(derivation) level* 0. For every $k \geqslant 1$, the *chase of level up to* k of D relative to Σ, denoted $\mathit{chase}^k(D, \Sigma)$, is constructed as follows: let $I_1, \ldots, I_n$ be all possible images of bodies of TGDs in Σ relative to some homomorphism such that (i) $I_1, \ldots, I_n \subseteq \mathit{chase}^{k-1}(D, \Sigma)$ and (ii) the highest level of an atom in every I_i is $k-1$; then, perform every corresponding TGD application on $\mathit{chase}^{k-1}(D, \Sigma)$, choosing the applied TGDs and homomorphisms in a (fixed) linear and lexicographic order, respectively, and assigning to every new atom the *(derivation) level* k. The *chase* of D relative to Σ, denoted $\mathit{chase}(D, \Sigma)$, is then defined as the limit of $\mathit{chase}^k(D, \Sigma)$ for $k \rightarrow \infty$.

The (possibly infinite) chase relative to TGDs is a *universal model*, i.e., there exists a homomorphism from $\mathit{chase}(D, \Sigma)$ onto every $B \in \mathit{mods}(D, \Sigma)$ [3, 7]. This result implies that BCQs Q over D and Σ can be evaluated on the chase for D and Σ, i.e., $D \cup \Sigma \models Q$ is equivalent to $\mathit{chase}(D, \Sigma) \models Q$. In the case of guarded TGDs Σ, such BCQs Q can be evaluated on an initial fragment of $\mathit{chase}(D, \Sigma) \models Q$ of constant depth $k \cdot |Q|$, and thus be done in polynomial time in the data complexity.

Note that sets of guarded TGDs (with single-atom heads) are theories in the guarded fragment of first-order logic [1]. Note also that guardedness is a truly fundamental class ensuring decidability as adding a single unguarded Datalog rule to a guarded Datalog$^\pm$ program may destroy decidability as shown in [3].

2.4 Negative Constraints

Another crucial ingredient of Datalog$^\pm$ for ontological modeling are *negative constraints (NCs*, or simply *constraints*), which are first-order formulas of the form $\forall \mathbf{X}\, \Phi(\mathbf{X}) \rightarrow \bot$, where $\Phi(\mathbf{X})$ is a conjunction of atoms (not necessarily guarded). We usually omit the universal quantifiers, and we implicitly assume that all sets of constraints are finite here. Adding negative constraints to answering BCQs Q over databases and guarded TGDs is computationally easy, as for each constraint $\forall \mathbf{X} \Phi(\mathbf{X}) \rightarrow \bot$, we only have to check that the BCQ $\Phi(\mathbf{X})$ evaluates to false; if one of these checks fails, then the answer to the original BCQ Q is positive, otherwise the negative constraints can be simply ignored when answering the original BCQ Q.

2.5 Equality-Generating Dependencies

A further important ingredient of Datalog$^{\pm}$ for modeling ontologies are *equality-generating dependencies* (or *EGDs*) σ, which are first-order formulas $\forall \mathbf{X}\, \Phi(\mathbf{X}) \rightarrow X_i = X_j$, where $\Phi(\mathbf{X})$, called the *body* of σ, denoted $body(\sigma)$, is a (not necessarily guarded) conjunction of atoms, and X_i and X_j are variables from $\mathbf{X}$. We call $X_i = X_j$ the *head* of σ, denoted $head(\sigma)$. Such σ is satisfied in a database D for $\mathcal{R}$ iff, whenever there exists a homomorphism h such that $h(\Phi(\mathbf{X}, \mathbf{Y})) \subseteq D$, it holds that $h(X_i) = h(X_j)$. We usually omit the universal quantifiers in EGDs, and all sets of EGDs are finite here.

An EGD σ on $\mathcal{R}$ of the form $\Phi(\mathbf{X}) \rightarrow X_i = X_j$ is *applicable* to a database D for $\mathcal{R}$ iff there exists a homomorphism $\eta\colon \Phi(\mathbf{X}) \rightarrow D$ such that $\eta(X_i)$ and $\eta(X_j)$ are different and not both constants. If $\eta(X_i)$ and $\eta(X_j)$ are different constants in Δ, then there is a *hard violation* of σ (and, as we will see below, the *chase* fails). Otherwise, the result of the application of σ to D is the database $h(D)$ obtained from D by replacing every occurrence of a non-constant element $e \in \{\eta(X_i), \eta(X_j)\}$ in D by the other element e' (if e and e' are both nulls, then e precedes e' in the lexicographic order). The *chase* of a database D, in the presence of two sets Σ_T and Σ_E of TGDs and EGDs, respectively, denoted $chase(D, \Sigma_T \cup \Sigma_E)$, is computed by iteratively applying (1) a single TGD once, according to the standard order and (2) the EGDs, as long as they are applicable (i.e., until a fixpoint is reached). To assure that adding EGDs to answering BCQs Q over databases and guarded TGDs along with negative constraints does not increase the complexity of query answering, all EGDs are assumed to be *separable* [2]. Intuitively, separability holds whenever: (i) if there is a hard violation of an EGD in the chase, then there is also one on the database w.r.t. the set of EGDs alone (i.e., without considering the TGDs); and (ii) if there is no chase failure, then the answers to a BCQ w.r.t. the entire set of dependencies equals those w.r.t. the TGDs alone (i.e., without the EGDs).

2.6 Guarded Datalog$^{\pm}$ Ontologies

We define (guarded) Datalog$^{\pm}$ ontologies as follows. A *(guarded) Datalog$^{\pm}$ ontology* consists of a database D, a (finite) set of guarded TGDs Σ_T, a (finite) set of negative constraints Σ_C, and a (finite) set of EGDs Σ_E that are separable from Σ_T.

3 Probabilistic Datalog$^{\pm}$

We consider a probabilistic extension of Datalog$^{\pm}$ based on Markov logic networks (MLNs) [19] as introduced in [11]. We now briefly recall its syntax and semantics.

3.1 Syntax

We assume an infinite universe of data constants Δ, an infinite set of labeled nulls Δ_N, and an infinite set of variables $\mathcal{V}$, as in Datalog$^{\pm}$. Furthermore, we assume a finite set of random variables X, as in MLNs. Informally, a probabilistic guarded Datalog$^{\pm}$ ontology consists of a finite set of probabilistic atoms, guarded TGDs, negative constraints, and separable EGDs, along with an MLN. We provide the formal details next.

We first define the notion of probabilistic scenario. A *(probabilistic) scenario* λ is a (finite) set of pairs (X_i, x_i), where $X_i \in X$, $x_i \in Dom(X_i)$, and the X_i's are pairwise distinct. If $|\lambda| = |X|$, then λ is a *full* probabilistic scenario. If every random variable X_i has a Boolean domain, then we also abbreviate λ by the set of all X_i such that $(X_i, true) \in \lambda$. Intuitively, a probabilistic scenario is used to describe an event in which the random variables in an MLN are compatible with the settings of the random variables described by λ, i.e., each X_i has the value x_i.

If a is an atom, σ_T is a TGD, σ_C is a negative constraint, σ_E is an EGD, and λ is a probabilistic scenario, then: (i) $a\colon \lambda$ is a *probabilistic atom*; (ii) $\sigma_T\colon \lambda$ is a *probabilistic TGD (pTGD)*; (iii) $\sigma_C\colon \lambda$ is a *probabilistic (negative) constraint*; and (iv) $\sigma_E\colon \lambda$ is a *probabilistic EGD (pEGD)*. We also refer to probabilistic atoms, TGDs, (negative) constraints, and EGDs as *annotated formulas*. Intuitively, annotated formulas hold whenever the events associated with their probabilistic scenarios occur.

A *probabilistic (guarded) Datalog$^\pm$ ontology* is a pair $\Phi = (O, M)$, where O is a finite set of probabilistic atoms, guarded TGDs, constraints, and EGDs, and M is an MLN. In the sequel, we implicitly assume that every such $\Phi = (O, M)$ is *separable*, which means that Σ_E^ν is separable from Σ_T^ν, for every $\nu \in Dom(X)$, where Σ_T^ν (resp., Σ_E^ν) is the set of all TGDs (resp., EGDs) σ such that (i) $\sigma\colon \lambda \in O$ and (ii) λ is contained in the set of all $(X_i, \nu(X_i))$ with $X_i \in X$. As for queries, we are especially interested in the probabilities of the answers of CQs to probabilistic Datalog$^\pm$ ontologies, called *probabilistic conjunctive queries* (PCQs).

3.2 Semantics

The semantics of probabilistic Datalog$^\pm$ ontologies is given relative to probability distributions over *interpretations* $\mathcal{I} = (D, \nu)$, where D is a database, and $\nu \in Dom(X)$. We say $\mathcal{I}$ *satisfies* an annotated formula $F\colon \lambda$, denoted $\mathcal{I} \models F\colon \lambda$, iff whenever $\nu(X) = x$, for all $(X, x) \in \lambda$, then $D \models F$. A *probabilistic interpretation* is a probability distribution Pr over the set of all possible interpretations such that only a finite number of interpretations are mapped to a non-zero value. The probability of an annotated formula $F\colon \lambda$, denoted $Pr(F\colon \lambda)$, is the sum of all $Pr(\mathcal{I})$ such that $\mathcal{I} \models F\colon \lambda$.

Let Pr be a probabilistic interpretation, and $F\colon \lambda$ be an annotated formula. We say that Pr *satisfies* (or is a *model* of) $F\colon \lambda$ iff $Pr(F\colon \lambda) = 1$. Furthermore, Pr is a model of a probabilistic Datalog$^\pm$ ontology $\Phi = (O, M)$ iff: (i) Pr satisfies all annotated formulas in O, and (ii) $1 - Pr(\mathit{false}\colon \lambda) = Pr_M(\lambda)$ for all full probabilistic scenarios λ, where $Pr_M(\lambda)$ is the probability of $\bigwedge_{(X_i, x_i) \in \lambda}(X_i = x_i)$ in the MLN M (and computed in the same way as $P(X = x)$ in MLNs).

As for the semantics of queries, we begin with defining the semantics of PCQs without free variables. Let Φ be a probabilistic Datalog$^\pm$ ontology, and Q be a BCQ. The *probability* of Q in Φ, denoted $Pr^{\Phi}(Q)$, is the infimum of $Pr(Q : \{\})$ subject to all probabilistic interpretations Pr such that $Pr \models \Phi$. Note that, as a consequence, the probability of a BCQ Q is the sum of all probabilities of full scenarios where the resulting universal model satisfies Q. We next consider the general case. As usual, given a set of variables V and a set of constants Δ, a *substitution* of V by Δ is a mapping $\theta\colon V \to \Delta$; given a formula F and substitution θ, we denote by $F\theta$ the formula obtained from F by replacing all variables v_i with $\theta(v_i)$. We can now define answers to PCQs. Let Φ be a probabilistic Datalog$^\pm$ ontology, and Q be a CQ. An *answer* for Q to Φ is a pair (θ, p), where (i) θ is a substitution for the free variables of Q, and (ii) $p \in [0, 1]$ is the probability of $Q\theta$ in Φ. It is *positive* iff $p > 0$.

4 Ontology Mappings with Datalog$^\pm$

As a language integrating the description logics and the logic programming paradigm with TGDs, Datalog$^\pm$ allows to nicely tie together the theoretical results on information integration in databases and the work on ontology mediation in the Semantic Web.

When integrating data stored in databases or data warehouses, i.e., data organized by database schemas, usually so-called *source-to-target TGDs (s-t TGDs)*, corresponding to so-called *GLAV (global-local-as-view) dependencies*, are used as mappings.

According to [9], a schema mapping is defined as $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma_{st}, \Sigma_t)$, where **S** and **T** are the source and the target schema, respectively, Σ_{st} is the set of source-to-target TGDs and EGDs, and Σ_t is the set of target TGDs and EGDs, respectively.

The following two types of dependencies are important special cases of source-to-target TGDs: LAV (local-as-view) and GAV (global as view) as explained below:

- A **LAV (local as view)** dependency is a source-to-target TGD with a single atom in the body, i.e., it has the form $\forall \mathbf{X}\ A_S(\mathbf{X}) \rightarrow \exists \mathbf{Y} \psi(\mathbf{X}, \mathbf{Y}))$, where A_S is an atom over the source schema, and $\psi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over the target schema.
- A **GAV (global as view)** dependency is a source-to-target TGD with a single atom in the head, i.e., of the form $\forall \mathbf{X}\ \phi(\mathbf{X}) \rightarrow A_T(\mathbf{X}')$, where $\phi(\mathbf{X})$ is a conjunction of atoms over the source schema, and $A_T(\mathbf{X}')$ is an atom over the target schema with $\mathbf{X}' \subseteq \mathbf{X}$.

The following mappings that are mentioned in [4] as essential can also be represented in Datalog$^\pm$:

- **Copy (Nicknaming):** Copy a source relation or role (of arbitrary arity n) into a target relation or role (of the same arity n like the source relation or role) and rename it. Note that this kind of mapping is a LAV and a GAV mapping at the same time. Example[1]:
$$\forall x, y\ S\text{:location}(x, y) \rightarrow T\text{:address}(x, y).$$
- **Projection (Column Deletion):** Create a target relation or concept or role by deleting one or more columns of a source relation or source concept or source role (of arbitrary arity $n \geq 2$). Note that this kind of mapping is a LAV and GAV mapping at the same time. Example:
$$\forall x, y\ S\text{:author}(x, y) \rightarrow T\text{:person}(x).$$
- **Augmentation (Column Addition):** Create a target relation or role (of arbitrary arity $n \geq 2$) by adding one or more columns to the source relation or role or concept. Note that this is a LAV dependency. Example:

[1] Note that all examples are stemming from a consideration of the OAEI benchmark set, more specifically, the ontologies 101 and 301-303.

$$\forall x\ S\text{:editor}(x) \rightarrow \exists z\ T\text{:hasEditor}(z, x).$$

- **Decomposition:** Decompose a source relation or source role (of arbitrary arity n) into two or more target relations or roles or concepts. Note that this is a LAV dependency. Example:

$$\forall x, y\ S\text{:publisher}(x, y) \rightarrow T\text{:organization}(x), T\text{:proceedings}(y).$$

Only one mapping construct mentioned in [4] as essential cannot be represented by Datalog$^{\pm}$, and this is the join. As each TGD has to be guarded, there must be an atom in the body that contains all non-existentially quantified variables and, hence, a join like $\forall x, y\ S\text{:book}(y), S\text{:person}(x) \rightarrow T\text{:author}(x, y)$ cannot be represented with Datalog$^{\pm}$.

In ontology mediation, the definition of a *mapping* or *alignment* is based on correspondences between so-called matchable entities of two ontologies. The following definition is based on [8]: Let S and T be two ontologies that are to be mapped onto each other; let q be a function that defines the sets of matchable entities $q(S)$ and $q(T)$. Then, a correspondence between S and T is a triple $\langle e_1, e_2, r\rangle$ with $e_1 \in q(S)$, $e_2 \in q(T)$ and r being a semantic relation between the two matchable elements. A *mapping* or *alignment* between S and T then is a set of correspondences $C = \cup_{i,j,k}\{\langle e_i, e_j, r_k\rangle\}$ between S and T. Note that this is a very general definition that basically allows to describe any kind of mapping language.

Semantic Web and ontology mapping languages usually contain a subset of the above mentioned mapping expressions and in addition constraints, mainly class disjointness constraints as additional mapping expressions (see also [21, 18]). However, note that both research communities, the data integration and the ontology mediation community, also proposed mapping languages that are also more expressive than even the above mentioned normal source-to-target TGDs, e.g., second-order mappings as described in the requirements of [20] or second-order TGDs [10]. In [18], a probabilistic mapping language based on MLNs that is built by mappings of a couple of basic description logic axioms onto predicates with the desired semantics has been presented. A closer look reveals that the mapping constructs that are used are renaming, decomposition and class disjointness constraints, and combinations thereof.

With Datalog$^{\pm}$, such disjointness constraints can be modeled with NCs Σ_{NC}:

- **Disjointness of ontology entities with the same arity:** A source relation (or role or concept) with arity n is disjoint to another relation (or role or concept) with the same arity n. The NC below corresponds to class disjointness that specifies that persons cannot be addresses:

$$\forall x\ S\text{:Person}(x), T\text{:Address}(x) \rightarrow \bot.$$

- **Disjointness of ontology entities with different arity:** A source relation (or role) with arity $n \geq 2$ is disjoint to another relation (or role or concept) with the arity $n > m \geq 1$. The example below specifies that persons do not have prices.

$$\forall x, y\ S\text{:Person}(x), T\text{:hasPrice}(x, y) \rightarrow \bot.$$

EGDs are also part of some mapping languages, especially in the database area, and can be represented by Datalog$^{\pm}$ as long as they are separable from the TGDs. Such kinds of dependencies allow to create mappings like the one of the following form specifying that publishers of the same book or journal in both, the source and target schema (or ontology), have to be the same:

$$\forall x, y, z\; S\text{:publisher}(x, y), T\text{:publishes}(y, z) \rightarrow x = z.$$

5 Ontology Mappings with Probabilistic Datalog$^{\pm}$

A *probabilistic (guarded) Datalog*$^{\pm}$ *mapping* has the form $\mathcal{M} = (\mathbf{S}, \mathbf{T}, p\Sigma_{st}, p\Sigma_t, M)$, where (i) **S** and **T** are the source and the target schemas or ontologies, respectively, (ii) $p\Sigma_{st}$ is a set of probabilistic (guarded) TGDs, EGDs, and NCs encoding the probabilistic source-to-target dependencies, (iii) $p\Sigma_t$ is a set of probabilistic (guarded) TGDs, EGDs, and NCs encoding the probabilistic target dependencies, and (iv) M is the MLN encoding the probabilistic worlds.

Observe here that the TGDs, EGDs, and NCs are annotated with probabilistic scenarios λ that correspond to the worlds that they are valid in. The probabilistic dependencies that the annotations are involved in are represented by the MLN. As annotations cannot refer to elements of the ontologies or the mapping except of the MLN itself, there is a modeling advantage of separating the two tasks of ontology modeling and of modeling the uncertainty around the axioms of the ontology.

Note that due to the disconnected representation between the probabilistic dependencies and the ontology, we can encode part of mapping formulas as predicates encoding a specific semantics like disjointness, renaming, or decomposition, in a similar way as done in [18]. With these predicates, an MLN can be created and the actual mappings can be enriched by ground predicates that add the probabilistic interpretation.

However, another more interesting encoding consists of using a second ontology describing additional features of the generation of the mappings and in this way eventually even do meta reasoning about the mapping generation. A rather general example of such an additional MLN describing the generation of a mapping is shown in Fig. 1. In this example, the MLN describes the generation of a mapping by means of the matcher that it generates and a set of — possibly dependent — applicability conditions as well as additional conditions that influence the probability of the mapping besides the result of the matcher.

With such kind of an MLN describing the dependency of different kinds of conditions (also dependencies between matchers are conceivable in order to combine the results of several different matchers), probabilistic reasoning over data integration settings can be done in more precise settings. To our knowledge, such kinds of probabilistic meta ontologies for the matching process have not yet been proposed.

6 Provenance

Data provenance information describes the history of data in its life cycle. It adds value to the data by explaining how it was obtained. In information integration, when data from distributed databases or ontologies is integrated, provenance information allows to check the trustworthiness and correctness of the results of queries and debug them

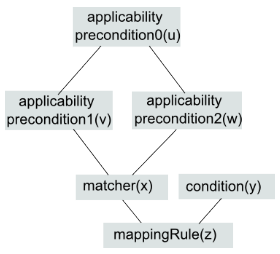

applicability precondition0(u)
applicability precondition1(v)
applicability precondition2(w)
matcher(x)
condition(y)
mappingRule(z)

Fig. 1. Example of an MLN describing the generation of mappings by means of applicability conditions and an additional condition that influences the probability of the mapping besides the result of the matcher.

as well as trace the errors back to where they have been created. Hence, an information integration framework should be equipped by some form of provenance.

In data provenance, it is mainly distinguished between where-, why- and how-provenance [5]. How-provenance [12] is the most expressive one and most appropriate for our purpose of annotating mappings and tracing back the origin of query results. How-provenance is modeled by means of a semiring. It is possible to construct different kinds of semirings depending on what kind of information has to be captured and which operations on that information are to be allowed. Besides formalizing different kinds of provenance annotations with a certain kind of semiring (called K-relations) based on the positive relational algebra, [12] provides a formalization of plain Datalog without negation with K-relations that is used within the collaborative data sharing system ORCHESTRA [13] also for modeling TGDs without existential quantifiers. In order to capture applications of mappings in ORCHESTRA, [15] proposes to use a so-called $\mathcal{M}$-semiring, which allows to annotate the mappings with $\mathcal{M} = m_1, \ldots, m_k$ being a set of mapping names, which are unary functions, one for each mapping. This can be combined with the formalization of negation-free Datalog (with a procedural semantics based on the least fixpoint operator to construct the model) with positive K-relations as presented in [12].

Clearly, such kind of a formalization for our probabilistic Datalog$^{\pm}$ information integration framework would allow to capture provenance and annotate the mappings with an id such that the integration paths can be traced back to their origin. In this way, routes that can be used to debug mappings like in [6] can be captured. In addition, as shown in [12], when the mappings are the only probabilistic or uncertain elements, the probabilities can also be computed more efficiently as the captured provenance also carries the information where the probabilities are propagated from. In addition, cycles can be detected and the trustworthiness of query results can also be estimated, as it can be detected where the data that is involved in the query result has been integrated from. For this purpose, the trustworthiness of data sets and possibly also peers who provide access to data sets need to be assessed beforehand.

In order to use a similar approach as the aforementioned ORCHESTRA system, we need to investigate how to model the application of the chase within probabilistic Datalog$^{\pm}$ with a semiring formalization. It can be expected that in probabilistic data integration with Datalog$^{\pm}$, the lineage will be restricted by the guards who help to direct the chase towards the answer of a query through the annotated guarded chase forest.

7 Summary and Outlook

By means of probabilistic (guarded) Datalog$^{\pm}$ [11], which can represent *DL-Lite* and $\mathcal{EL}$, we use a tractable language with dependencies that allows to nicely tie together the theoretical results on information integration in databases and the work on ontology mediation in the Semantic Web. The separation between the ontology and the probabilistic dependencies allows us to either model the mappings with specific newly invented predicates like disjointness, renaming, or decomposition, etc. or — more interestingly — with a probabilistic meta ontology describing the matching process.

The paper shows how classical and probabilistic (guarded) Datalog$^{\pm}$ can be used to model information integration settings and sketches a deterministic mapping language based on Datalog$^{\pm}$ and two different kinds of probabilistic adaptations based on the rather loosely coupled probabilistic extension of Datalog$^{\pm}$ with worlds represented by means of an MLN. We also justify why data provenance needs to be captured and represented within such a probabilistic information integration framework and propose to use an adaptation of K-relations as proposed by [12]. Such an extension with provenance allows to track how results of queries to the framework have been created and also debug mappings as errors can be traced back to their origin.

As a next step, we will develop the proposed framework for provenance capture and, amongst others, investigate how to model the chase application for reasoning with probabilistic (guarded) Datalog$^{\pm}$ with a semiring-framework.

Acknowledgments. This work was supported by an EU (FP7/2007-2013) Marie-Curie Intra-European Fellowship, the Engineering and Physical Sciences Research Council (EPSRC) grant EP/J008346/1 "PrOQAW: Probabilistic Ontological Query Answering on the Web", the European Research Council (FP7/2007-2013/ERC) grant 246858 ("DIADEM"), and by a Yahoo! Research Fellowship.

References

1. Andréka, H., Németi, I., van Benthem, J.: Modal languages and bounded fragments of predicate logic. Journal of Philosophical Logic 27(3), 217–274 (1998)
2. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. Journal of Web Semantics 14, 57–83 (2012)
3. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive integrity constraints. In: Proceedings KR-2008, pp. 70–80. AAAI Press (2008)
4. ten Cate, B., Kolaitis, P.G.: Structural characterizations of schema-mapping languages. Communications of the ACM 53, 101–110 (2010)
5. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how and where. Foundation and Trends in Databases 1, 379–474 (2009)
6. Chiticariu, L., Tan, W.C.: Debugging schema mappings with routes. In: Proceedings VLDB-2006, pp. 79–90. ACM Press (2006)

7. Deutsch, A., Nash, A., Remmel, J.: The chase revisited. In: Proceedings PODS-2008, pp. 149–158. ACM Press (2008)
8. Euzenat, J., Shvaiko, P.: Ontology Matching. Springer (2007)
9. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theoretical Computer Science 336(1), 89–124 (2005)
10. Fagin, R., Kolaitis, P.G., Popa, L., Tan, W.C.: Composing schema mappings: Second-order dependencies to the rescue. ACM Transactions on Database Systems 30, 994–1055 (2005)
11. Gottlob, G., Lukasiewicz, T., Martinez, M.V., Simari, G.I.: Query answering under probabilistic uncertainty in Datalog+/– ontologies. Annals of Mathematics and Artificial Intelligence (2013)
12. Green, T.J., Karvounarakis, G., Tannen, V.: Provenance semirings. In: Proceedings PODS-2007, pp. 31–40. ACM Press (2007)
13. Green, T.J., Karvounarakis, G., Taylor, N.E., Biton, O., Ives, Z.G., Tannen, V.: ORCHESTRA: Facilitating collaborative data sharing. In: Proceedings SIGMOD-2007, pp. 1131–1133. ACM Press (2007)
14. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. Journal of Computer and System Sciences 28(1), 167–189 (1984)
15. Karvounarakis, G.: Provenance in collaborative data sharing. Ph.D. thesis, University of Pennsylvania (2009)
16. Lenzerini, M.: Data integration: A theoretical perspective. In: Proceedings PODS-2002, pp. 233–246. ACM Press (2002)
17. Maier, D., Mendelzon, A.O., Sagiv, Y.: Testing implications of data dependencies. ACM Transactions on Database Systems (TODS) 4(4), 455–469 (1979)
18. Niepert, M., Noessner, J., Meilicke, C., Stuckenschmidt, H.: Probabilistic Logical Web Data Integration, chap. Reasoning Web. Semantic Technologies for the Web of Data, pp. 504–533. Springer (2011)
19. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning 62(1/2), 107–136 (2006)
20. Scharffe, F., de Bruijn, J.: A language to specify mappings between ontologies. In: Proceedings SITIS-2005, pp. 267–271. IEEE Computer Society (2005)
21. Serafini, L., Stuckenschmidt, H., Wache, H.: A formal investigation of mapping language for terminological knowledge. In: Proceedings IJCAI-2005, pp. 576–581. Professional Book Center (2005)

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

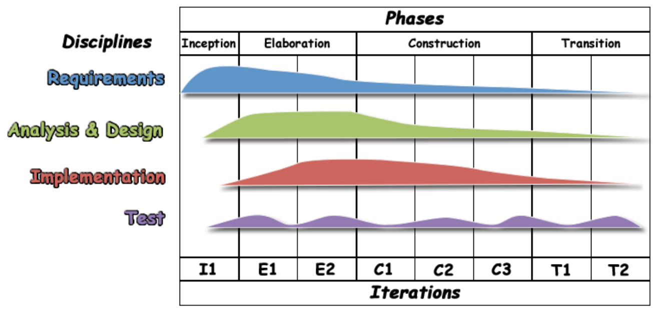


Fig. 1. Uncertainty Modeling Process for Semantic Technologies (UMP-ST).

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.

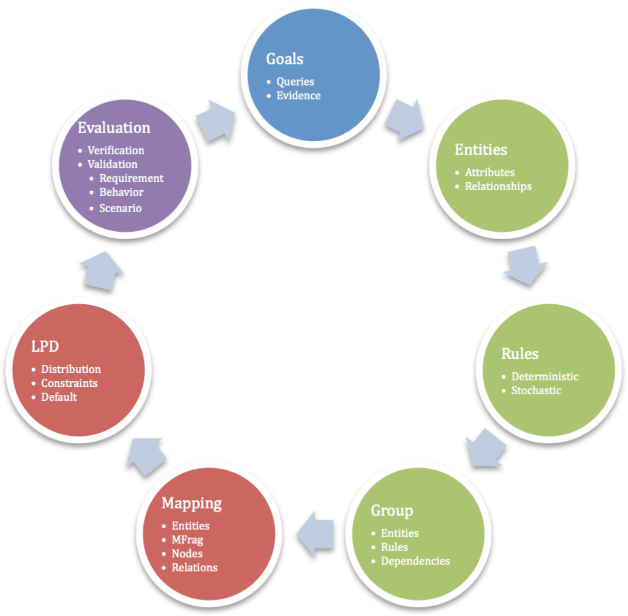


Fig. 2. Probabilistic Ontology Modeling Cycle (POMC) - Requirements in blue, Analysis & Design in green, Implementation in red, and Test in purple.

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$^{\text{TM}}$ 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 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.

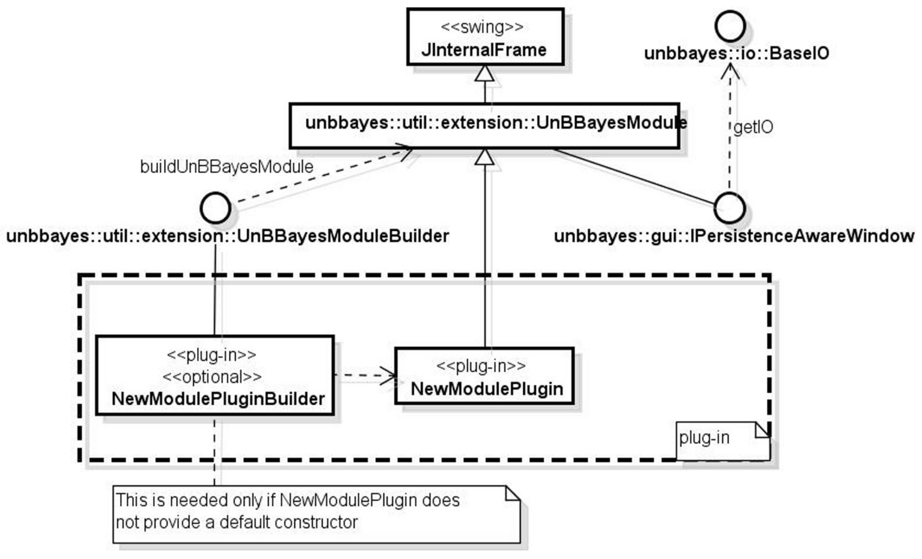
One specific type of plug-in that can be added to UnBBayes is the module plug-in. Module plug-ins provide a means to create a relatively self-sufficient feature in UnBBayes (*e.g.* new formalisms or completely new applications). In UnBBayes vocabulary, *modules* are basically new internal frames that 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.



<<swing>>
JInternalFrame
unbbayes::io::BaseIO
unbbayes::util::extension::UnBBayesModule
getIO
buildUnBBayesModule
unbbayes::util::extension::UnBBayesModuleBuilder
unbbayes::gui::IPersistenceAwareWindow
<<plug-in>>
<<optional>>
NewModulePluginBuilder
<<plug-in>>
NewModulePlugin
plug-in
This is needed only if NewModulePlugin does not provide a default constructor

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.

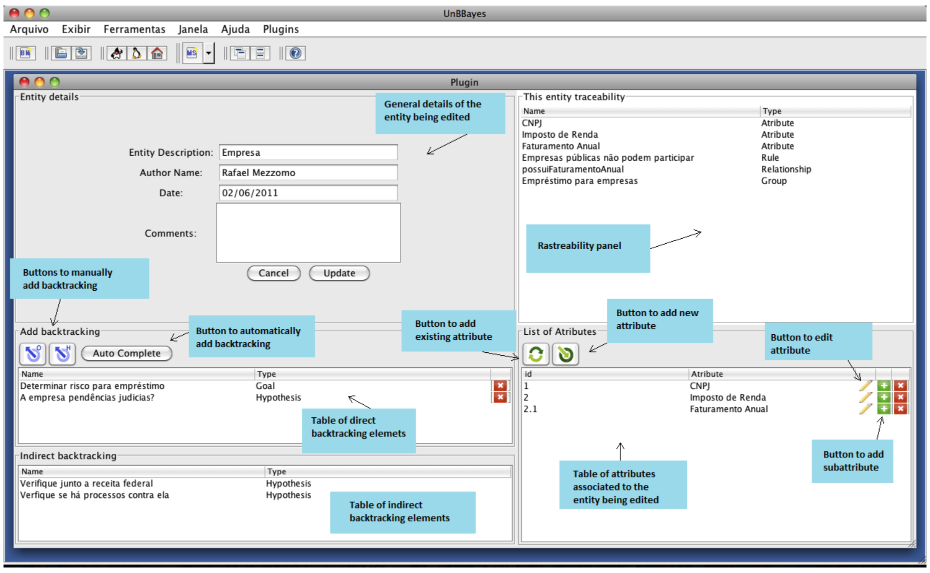
The UMP-ST plug-in is a almost like a wizard tool that guides the user in each and every step of the Requirements and Analysis & Design disciplines. This involves the definition of the goals that should be achieved by the probabilistic ontology (PO) as well as the queries that should be answered by the 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.


UnBBayes
Arquivo Exibir Ferramentas Janela Ajuda Plugins
Plugin
Entity details
General details of the entity being edited
Cancel
Update
Buttons to manually add backtracking
Add backtracking
Auto Complete
Button to automatically add backtracking
Button to add existing attribute
Table of direct backtracking elemets
Indirect backtracking
Table of indirect backtracking elements
This entity traceability
Rastreability panel
Button to add new attribute
List of Atributes
Button to edit attribute
Table of attributes associated to the entity being edited
Button to add subattribute

Entity Description:	Empresa
Author Name:	Rafael Mezzomo
Date:	02/06/2011
Comments:	

Name	Type
Determinar risco para empréstimo	Goal
A empresa pendências judicias?	Hypothesis

Name	Type
Verifique junto a receita federal	Hypothesis
Verfique se há processos contra ela	Hypothesis

Name	Type
CNPJ	Atribute
Imposto de Renda	Atribute
Faturamento Anual	Atribute
Empresas públicas não podem participar	Rule
possuiFaturamentoAnual	Relationship
Empréstimo para empresas	Group

id	Atribute
1	CNPJ
2	Imposto de Renda
2.1	Faturamento Anual

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:

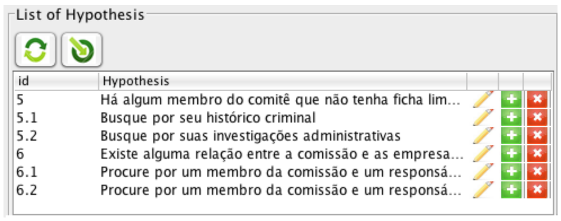
1. *Goal*: Identify whether the committee of a given procurement should be changed.
   (a) *Query*: Is there any member of committee who does not have a clean history?
      i. *Evidence*: Committee member has criminal history;
      ii. *Evidence*: Committee member has been subject to administrative 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.


List of Hypothesis

id	Hypothesis
5	Há algum membro do comitê que não tenha ficha lim...
5.1	Busque por seu histórico criminal
5.2	Busque por suas investigações administrativas
6	Existe alguma relação entre a comissão e as empresa...
6.1	Procure por um membro da comissão e um responsá...
6.2	Procure por um membro da comissão e um responsá...

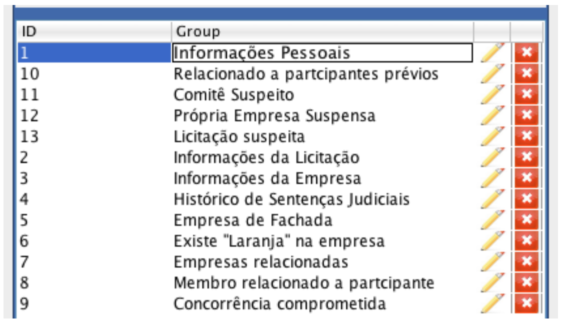
Fig. 5. Panel for displaying the hypothesis (queries and evidence) for a goal.

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.

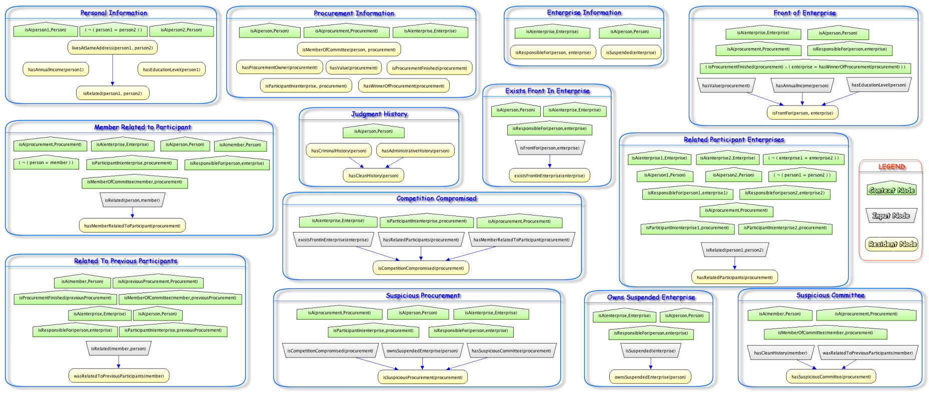


Fig. 7. Implementation of the PO in UnBBayes-MEBN.

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.

Position Papers

Handling uncertainty in semantic information retrieval process

Chkiwa Mounira[1], Jedidi Anis[1] and Faiez Gargouri[1]

[1]Multimedia, InfoRmation systems and Advanced Computing Laboratory
Sfax University, Tunisia

m.chkiwa@gmail.com, jedidianis@gmail.com, faiez.gargouri@isimsf.rnu.tn

Abstract. This position paper proposes a collaboration method between Semantic Web and Fuzzy Logic aiming to handle uncertainty in the information retrieval process in order to cover more relevant items in result of search process. The collaboration method employs OWL ontology in query enhancement, RDF in annotation process and fuzzy rules in ranking enhancement.

1 Introduction

In the information retrieval process, there are returned documents which are relevant to the query but they focus in addition of query main interest on others additional topics. To deal with this imprecision we propose to valorize in the ranking process relevant documents which deal mainly with query themes. Another source of imprecision in the search process is the user queries; we propose to enhance it in order to come near the intention of the user. This paper is organized as follows: in the next section we present our proposition to enhance the query background expression then we explain how Semantic Web and Fuzzy Logic collaborate to enhance ranking process. In Section 3, we present some related works and Section 4 concludes the paper.

2 Handling uncertainty by semantic/fuzzy collaboration

2.1 The semantic/fuzzy query enhancement

A main cause of uncertainty in the information retrieval process comes from the user's queries. In order to return more relevant results, the information retrieval system has to indentify the user's intention behind the query. To do it, we propose to enhance user queries by adding semantically related terms. In this purpose, we use the Web Ontology Language OWL and then we employ some fuzzy rules in order to weight up the query terms importance. In Figure 1, we present our semantic/fuzzy query enhancement.

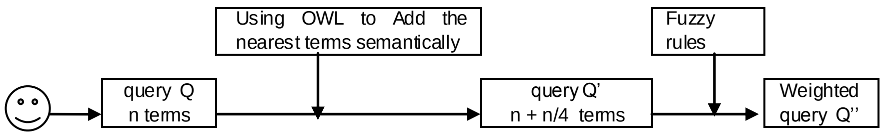


Fig. 1. Semantic/fuzzy query enhancement

The Semantic query enhancement passes through the enrichment of the query by new terms syntactically different but semantically near; the new added terms are not picked to derive the query meaning but to find terms expressing more the user intention. Several works as [1- 3] are proposed to express the semantic similarity between ontology concepts. After eliminating empty terms from the query, we can reuse the algorithm presented in [3] to find the semantically nearest term to each query term using OWL ontology. The number of added terms must not be constant; it can derive the query meaning if it is large or useless if it is few. So we decide that the number of added terms be proportional to the query length. Hence, we propose to add only n/4 terms having the highest similarity to query terms. Also, we propose that the information retrieval system is interactive and allows users to highlight certain query terms in order to reflect their importance. Finally, to weight the query terms, we apply some fuzzy rules; those rules define the priority of weighting:

- If a query term is added from the ontology then it will has low weight priority.
- If a query term is not bold, then it will has a medium weight priority.
- If a query term is bold then it will has a high weight priority.

2.2 The semantic/fuzzy ranking enhancement

The semantic/fuzzy ranking enhancement aims to manage uncertainty about the output of classic querying process and to valorize documents focusing specially in the same user query interests. It aims principally to limit the number of relevant documents dealing with several topics. The semantic/fuzzy ranking enhancement passes through two fundamental concepts: the "meta-document" which allows annotating semantically the collection of documents and the "themes clouds" which enhance the ranking process based on Fuzzy Logic. The meta-document is introduced in [4] and it is able to annotate semantically multimedia objects as well as web documents. A meta-document uses RDF metadata to annotate web resources in a way that ensures its reusability. The querying process matches the user query with the meta-documents in order to identify the score relevance of the resources to the query. We define the "theme cloud" as groups of weighted terms concerning a given theme. Simply, we collect potential terms representing a given theme to construct a theme cloud. The terms' weights express the ability of each term to represent the theme. After running a usual querying process matching the query and the meta-documents, we get the relevance score for each annotated resource or document. At this point, the theme clouds are used to enhance ranking results in the benefit of relevant documents focusing mainly on query interests. The Figure 2 gives a simple presentation of the structure of the semantic/fuzzy ranking enhancement:

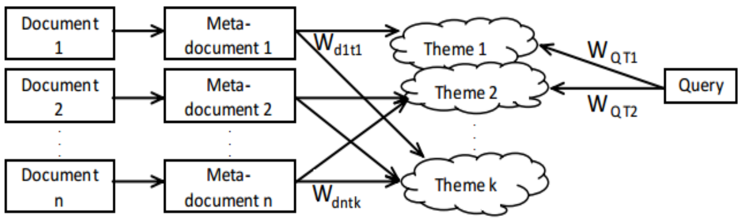


Fig. 2. The semantic/fuzzy ranking enhancement

To run the ranking enhancement, first, we establish the meta-document/theme weighted links $W_{DT.}$ W_{DT} expresses the potential themes mentioned by the meta-document. To assign a weight W_{DT} to a meta-document/theme link, we simply sum the weights of theme terms existing in the meta-document. Then we establish query/themes weighted links which express the ability of each theme to represent the query. To assign a weight to a query/theme link, we use the classic similarity measure between two weighted terms vectors:

$$W_{QT} = sim(Q,Ti) = \frac{\sum_{j=1}^{t} W_{qj} * W_{t_{ij}}}{\sqrt{\sum_{j=1}^{t} (W_{qj})^2 * \sum_{j=1}^{t} (W_{t_{ij}})^2}} \qquad (1)$$

The next step of ranking enhancement is to calculate for each document his theme similarity with the query in order to increase or decrease its relevance score in terms of the value of the theme similarity. The theme similarity TS is calculated as follow:

$$TS\ (Q,D) = \sum_{i=1}^{k} |W_{QT_i} - W_{DT_i}| \qquad (2)$$

The main goal of the ranking enhancement is to increase relevance of documents focusing on the same query themes and to decrease relevance of document dealing with different themes vis-à-vis the query. The TS (Q, D) value is optimal when its value is minimal; this means that the query and the document are focusing on the same themes with approached values. Contrariwise, if the TS is high, this means that the document deals with other themes in addition to the query themes. Finally, the increase or the decrease Rate R affected to a document Relevance Score RS is based on the following fuzzy rules:

— If RS is high or medium and TS is low then R is high
— If RS is low and TS is low then R is medium
— If RS is low or medium and TS is high then R is negative

3 Related work

Several approaches considering both uncertainty and the Semantic Web have been proposed in the information retrieval issue. [5, 6] propose to fuzzify in different ways RDF triples, likewise [7, 8] propose to fuzzify OWL ontology statements. A common point in those works is the use of formal ways to express the assignment of a truth degree to RDF triples or OWL axioms. In our proposition, numerical membership values identification is done in background using mathematical deduction without the need of formal expressions (e.g. weight priority of a query term). Some other works are near our proposition: [9] shows that it is useful to express a fuzzy proximity values between terms of a query. By using a fuzzy set of rules [10] shows the usability of a ranking system based on fuzzy inference. In the query enhancement issue, many works are proposed [11-12]; our method is characterized by its simplicity and flexibility.

4 Conclusion

In this paper we studied two interoperable axes in the information retrieval process: the Semantic Web and the Fuzzy Logic. We propose to enhance query background expression and also to enhance ranking process using fuzzy rules. Given that Numerical inputs of fuzzy rules are deduced from the meta-documents characteristics, it remains to identify in the short run, the numerical limits to fuzzy sets on which we will apply the fuzzy rules set. Equally, we plan to extend the current proposition and to investigate the concept of user profile in order to cover more relevant result document.

5 References

1. B. Y. Liang, J. Tang, J. and Z. Li. Semantic Similarity Based Ontology Cache. Heidelberg: Springer-Verlag, 2006, pp.250-262.
2. Z. Yang. Semantic similarity measure matching between ontology concepts based on heuristic rules. Journal of computer applications, vol.27, no.12, pp.2919-2921, 2007.
3. Xiao Min; Zhong Luo; Xiong Qianxing. Semantic Similarity between Concepts Based on OWL Ontologies. 2nd International Workshop on Knowledge Discovery and Data Mining, 2009. WKDD 2009. vol., no., pp.749,752, 23-25 Jan. 2009.
4. Jedidi A. « modélisation générique de documents multimédia par des métadonnées : mécanismes d'annotation et d'interrogation » Thesis of « Université TOULOUSE III Paul Sabatier », France. July 2005.
5. Simou, N., Stoilos, G., Tzouvaras, V., Stamou, G., and Kollias ,S. Storing and Querying Fuzzy Knowledge in the Semantic Web. In Proceedings of the Fourth International Workshop on Uncertainty Reasoning for the Semantic Web, Karlsruhe, Germany,, 2008
6. Mazzieri, M., and Dragoni, A. F.. A fuzzy semantics for Semantic Web languages. In ISWC-URSW, pages 12-22. 2005

7. Calegari, S. and Ciucci, D. Fuzzy Ontology, Fuzzy Description Logics and Fuzzy-OWL. WILF '07 Proceedings of the 7th international workshop on Fuzzy Logic and Applications: Applications of Fuzzy Sets Theory Pages 118-126. 2007.
8. Bobillo F., Straccia U. Fuzzy Ontology Representation using OWL 2. International Journal of Approximate Reasoning 52(7):1073-1094, 2011.
9. Beigbeder, M. and Mercier, A. Application de la logique floue à un modèle de recherche d'information basé sur la proximité. In Actes 12th rencontres francophones sur la Logique Floue et ses Applications. Nantes, France, pp. 231-237. 2004.
10. Rubens, N. The application of Fuzzy Logic to the construction of the ranking function of information retrieval systems. Computer Modeling and New Technologies, Vol.10, No.1, 20-27. 2006.
11. Neda A., Latifur K. and Bhavani T. Optmized ontology-driven query expansion using map-reduce framework to facilitate federated queries. Comput. Syst. Sci. Eng. 27(2) (2012)
12. Min S, Il-Yeol S, Xiaohua H, Robert B. Allen. Integration of association rules and ontologies for semantic query expansion. Data Knowl. Eng. 63(1): 63-75 (2007).

Reliability Analyses of Open Government Data

Davide Ceolin[1], Luc Moreau[2] Kieron O'Hara[2], Guus Schreiber[1], Alistair Sackley[3], Wan Fokkink[1], Willem Robert van Hage[4], and Nigel Shadbolt[2]

[1] VU University Amsterdam, The Netherlands
{d.ceolin, guus.schreiber, w.j.fokkink}@vu.nl
[2] University of Southampton, United Kingdom
{l.moreau, kmo}@ecs.soton.ac.uk
[3] Hampshire County Council, United Kingdom
[4] SynerScope B.V., The Netherlands
willem.van.hage@synerscope.com

Abstract. Public authorities are increasingly sharing sets of open data. These data are often preprocessed (e.g. smoothened, aggregated) to avoid to expose sensible data, while trying to preserve their reliability. We present two procedures for tackling the lack of methods for measuring the open data reliability. The first procedure is based on a comparison between open and closed data, and the second derives reliability estimates from the analysis of open data only. We evaluate these two procedures over data from the `data.police.uk` website and from the Hampshire Police Constabulary in the UK. With the first procedure we show that the open data reliability is high despite preprocessing, while with the second one we show how it is possible to achieve interesting results concerning the open data reliability estimation when analyzing open data alone.

1 Introduction

Open Government Data are valuable for boosting the economy, enhancing the transparency of public administration and empowering the citizens. These data are often sensitive and so need to be preprocessed for privacy reasons. In the paper, we refer to the public Open Government Data as "open data" and to the original data as "closed data".

Different sources expose open data in different manners. For example, Crime Reports [4] and `data.police.uk` [15] both publish UK crime data, but in different format (maps vs. CSV files), level of aggregation, smoothing and timeliness (daily vs. monthly update), which all represent possible reasons for reliability variations. For different stakeholders it is important to understand how reliable different sources are. The police, who can access the closed data, needs to know if open data are reliable enough e.g. to be used in projects involving the citizens. The citizens wish to know the reliability of the different datasets to understand the reasons for differences between authoritative sources. We present two procedures to cope with the lack of methods to analyze these data: one for computing the reliability of open data by comparing them with the closed data, and one to estimate variations in the reliability of the open data by relying only on these.

The analysis of open data is spreading, led by the Open Data Institute (`http://www.theodi.org`) and others. For instance, Koch-Weser [10] presents an interesting analysis of the reliability of China's Economic Data, thus analyzing the same aspect as we are interested in, on a different typology of dataset. Tools for the quality estimation of open data are being developed (e.g. Talend Open Studio for Data Quality [14], Data Cleaner [8]), but their goal is less targeted than ours, since they aim at quantifying the quality of open data in general as to provide a substrate for a more comprehensive open data analysis infrastructure. Relevant for this work is also a paper from Ceolin et al. that uses a statistical approach to model categorical Web data [3] and one that uses provenance to estimate reliability [2]. We plan to adopt the approach proposed by Ebden et al. [6] to measure the impact of different processes on the data.

The rest of this paper is structured as follows: Section 2 describes a procedure for measuring the reliability of open data given closed data and a case study implementation; Section 3 presents a procedure for analyzing open data and a case study; lastly, Section 4 provides final discussion.

2 Procedure for Comparing Closed and Open Data

The UK Police Home Office aggregates (i.e., presents coarsely) and smoothens (introduces some small error) the open data for privacy reasons. We represent the open data provenance with the PROV Ontology [17] as in Fig. 1. In general, a faulty aggregation process or aggregating data coming from heterogeneous sources not properly manipulated might unexpectedly affect the resulting data reliability, while smoothing should affect it explicitly but in a limited and controlled manner. The following procedure aims at capturing such variation:

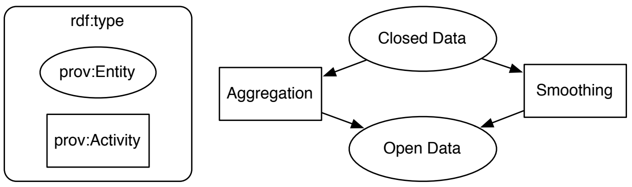


Fig. 1: Open Data Creation Provenance.

Select the relevant data Closed data might be spurious, so we select the data items that are relevant for our analyses. The selection of the data might involve the temporal aspect (i.e. only data referring to the relevant period are considered), their geographical location (select only the data regarding the area of interest), or other constraints and their combination;

Roll up categorical data There exists a hierarchy of categories because each level is available to a different audience: open data are presented coarsely to the citizens, while closed data are fine grained. We bring the categorization to the same level, hence bringing the closed data to the same level as the open data.

Compare the corresponding counts Different measures are possible, because the difference between datasets can be considered from different points of view: relative, absolute, etc.. For instance, the ratio of the correct items over the total amount or the Wilcoxon signed-rank test [18].

Case Study 1 We compare a set of crime counts per categories grouped per neighbourhood and month from `data.police.uk` with a limited set (30,436 relevant entries) of corresponding closed data from the Hampshire Constabulary by implementing the procedure above as follows:

Data Selection Select the data for the relevant months and geographical area. In this latter case, we load the KML file describing the Hampshire Constabulary area using the maptools library [1] in the R environment [13] and check if the crimes coordinates occur therein using the SDMTools library [16];

Data Aggregation We apply two kinds of aggregation: **temporal**, to group together data about the same month and **geographical**, to aggregate per neighbourhood. The closed data items report the address of occurrence of the crimes, while the open data are aggregated per police neighbourhood. We match the zip code of the addresses and the neighbourhoods using the MapIt API [11].

Data Comparison We average the result of the Wilcoxon signed-rank test applied per neighbourhood, to compare open and aggregated closed data.

For each neighbourhood we compute a Wilcoxon signed-rank test to check the significance of the difference between open and closed data and we average the outcomes (see Table 1a). We compute the test on the differences of the two counts (open and closed data) to check whether the estimated average of the distribution of the differences is zero (that is, the two distributions are statistically equivalent) or not.

The results at our disposal are limited, since we could analyze only two complete months. Still, we can say that smoothing, in these datasets, introduces a small but significant error. The highest error average (2.75) occurs with the entry with the highest error variance: this suggests that the higher error is due to a few, sparse elements, and not to the majority of the items. To prove this, we checked the error distribution among the entities and we reported the results in Table 1b. A χ^2 test [12] at 95% confidence level confirms that the two error distributions do not differ in a statistically significant manner.

3 Procedure for Analyzing Open Data

We propose here a procedure for analyzing open data alone, to be used when closed data are not available, which provides weaker but still useful results,

Table 1: Statistics about the errors in the comparison between open and closed data, and error distibution.

(a) Statistics about the comparison of open and closed data.

Months	Avg error	Var error	% Different Entries
month_1	2.75	12.28	79%
month_2	0.86	3.52	86%

(b) Percentage of items in each open dataset presenting a relative error of at most 0%, 25%, 50%, 75% and 100% with respect to the corresponding closed data item.

Month	% of Entries per Relative Error				
	0%	$\leq$ 25%	$\leq$ 50%	$\leq$ 75%	$\leq$ 100%
month_1	35%	44%	65%	74%	96%
month_2	34%	43%	57%	65%	91%

compared to the previous one. It compares each dataset with the consecutive one, measures their similarity and pinpoints the occurrence of possible reliability changes based on variations of similarity over time. We use a new similarity measure for comparing datasets, that aggregates different similarity "tests" performed on couples of datasets. Given two datasets d_1 and d_2, their similarity is computed as follows:

$$sim(d_1, d_2) = avg(t_1(d_1, d_2), \ldots, t_n(d_1, d_2))$$

where avg aggregates the results of n similarity tests t_i, with $i \in \{1 \ldots n\}$. We propose the following families of tests, although we are not restricted to them:

Statistical test Check with a statistical test (e.g. Wilcoxon signed-rank test) if the data are drawn from significantly different distributions.

Model Comparison test Build a model (e.g. linear regression [7] or Support Vector Machines [5]) on one of the two datasets and evaluate its performance (precision, recall) over the other dataset. These models represent an abstraction over the first dataset and by evaluating them over the other one, we check, according to such a model, how similar the two datasets are.

The tests can be aggregated, for instance, by averaging them or by merging them in a "subjective opinion" [9], which is a construct of a probabilistic logic that is equivalent to a Beta probability distribution about the correct value for the similarity. The expected value of the Beta is close to the arithmetical average, but the variance represents the uncertainty in our calculation, since it reduces as long as we consider more tests. The similarity measure alone does not stand for reliability: there can be many reasons for a similarity variation (e.g. a new law or a particular event that makes the crime rate rise) without implying a reliability change. Also, a similarity value alone might be difficult to interpret in terms of reliability, when a gold standard is not available. So we analyze the similarity of consecutive datasets to pinpoint items that possibly present reliability variations: if the similarity between datasets remains similar for a period of time, and then a variation occurs, one of the possible reasons for such a variation is a change in the data reliability. Unfortunately, we can not discriminate between this and other causes, unless we have additional information at our disposal.

Case Study 2 We analyze the police open data for the Hampshire Constabulary from `data.police.uk`, that consist of crime counts, aggregated per neighbourhood from April 2011 to December 2012. We know that in this period open data creation policy changes occurred. These might have affected the datasets reliability. We compare the distribution of the crime counts among the crime categories, and we represent the similarity between two datasets as the percentage of neighbourhoods that are statistically similar (according to a Wilcoxon signed-rank test). The results of the comparison are reported in Figure 2, where each point represents the similarity between two datasets, in sequence. At the twelfth comparison the similarity trend breaks and then starts a new one. That is likely to be a point where the reliability diverges as the similarity variation possibly hints, and it actually coincides with a policy change (the number of neighbourhoods varies from 248 to 232), and since the area divided by these neighbourhoods is the same, this possibly introduces a variation in the impact of the smoothing error, but we do not have at our disposal a confirmation of such impact. As we stressed earlier, the procedure allows us only to pinpoint possibly problematic data, but without additional information, our analysis cannot be precise, that is, we cannot be certain about the reason of the similarity change.

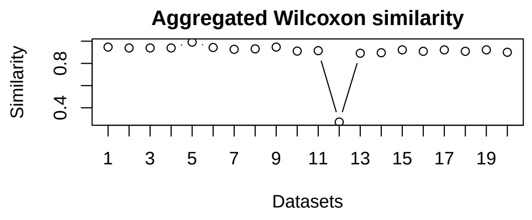


Fig. 2: Plot of the similarity of consequent datasets of crime counts for the Hampshire Constabulary from the `data.police.uk` website.

4 Conclusion and Future Work

We presented two procedures for the computation of the reliability of open data: one based on the comparison between open and closed data, the other one based on open data alone. Both procedures have been evaluated using data from the `data.police.uk` website and from the Hampshire Police Constabulary in the UK. The first procedure allows us to estimate the reliability of open data, and shows that smoothing procedures, although introducing some error, preserve a high data reliability. The second procedure is useful to grasp indications about the data reliability, although more weakly than the first one, since it allows only to pinpoint possible reliability variations in the data. Despite the fact that open data are exposed by authoritative institutions, these procedures allow us to enrich the open data with information about their reliability, to increase the confidence of both the insider specialist and the common citizen who use them and to help in understanding possible discrepancies between data exposed by different authorities. We plan to extend the range of analyses applied and of datasets considered. Moreover, we intend to map the data with Linked Data entities to combine the statistical analyses with semantics.

Acknowledgements This work is supported in part under SOCIAM: The Theory and Practice of Social Machines; the SOCIAM Project is funded by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant number EP/J017728/1.

References

1. R. Bivand. *Tools for reading and handling spatial objects*, 2013.
2. D. Ceolin, P. Groth, and W. R. van Hage. Calculating the Trust of Event Descriptions using Provenance. In *SWPM*, pages 11–16. CEUR-WS.org, Nov. 2010.
3. D. Ceolin, W. R. van Hage, W. Fokkink, and G. Schreiber. Estimating Uncertainty of Categorical Web Data. In *URSW*, pages 15–26. CEUR-WS.org, 2011.
4. CrimeReports. Crimereports. `https://www.crimereports.co.uk/`, July 2013.
5. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods.* Cambridge University Press, 2000.
6. M. Ebden, T. D. Huynh, L. Moreau, S. Ramchurn, and S. Roberts. Network analysis on provenance graphs from a crowdsourcing application. In *IPAW*, pages 168–182. Springer-Verlag, 2012.
7. F. Galton. Regression Towards Mediocrity in Hereditary Stature. *Journal of the Anthropological Institute*, 15:246–263, 1886.
8. Human Inference. DataCleaner. `http://datacleaner.org`, 2013.
9. A. Jøsang. A logic for uncertain probabilities. *Int. Journal Uncertainty Fuzziness Knowledge-Based Systems*, 9(3):279–311, June 2001.
10. I. N. Koch-Weser. The Reliability of China's Economic Data: An Analysis of National Output. `http://www.uscc.gov/sites/default/files/Research/TheReliabilityofChina'sEconomicData.pdf`, Jan. 2013.
11. mySociety. MapIt. `http://mapit.mysociety.orgs`, July 2013.
12. K. Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine*, 50:157–175, 1900.
13. R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Sept. 2012. ISBN 3-900051-07-0.
14. Talend. Talend Open Studio for Data Quality. `http://www.talend.com/products/data-quality`, 2013.
15. United Kingdom Police Home Office. data.police.uk. `data.police.uk`, July 2013.
16. J. VanDerWal, L. Falconi, S. Januchowski, L. Shoo, and C. Storlie. *SDMTools: Species Distribution Modelling Tools: Tools for processing data associated with species distribution modelling exercises*, 2012.
17. W3C. PROV-O: The PROV Ontology. `http://www.w3.org/TR/prov-o/`, July 2013.
18. F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

Towards Vagueness-Aware Semantic Data

Panos Alexopoulos[1], Boris Villazon-Terrazas[1], and Jeff Z. Pan[2]

[1] iSOCO, Intelligent Software Components S.A., Av. del Partenon, 16-18, 1-7, 28042, Madrid, Spain,
{palexopoulos,bvillazon}@isoco.com
[2] Department of Computing Science, University of Aberdeen, Meston Building, Aberdeen, AB24 3UE, UK.

Abstract. The emergence in recent years of initiatives like the Linked Open Data (LOD) has led to a significant increase in the amount of structured semantic data on the Web. In this paper we argue that the shareability and wider reuse of such data can very often be hampered by the existence of vagueness within it, as this makes the data's meaning less explicit. Moreover, as a way to reduce this problem, we propose a vagueness metaontology that may represent in an explicit way the nature and characteristics of vague elements within semantic data.

1 Introduction

Ontologies are formal shareable conceptualizations of domains, describing the meaning of domain aspects in a common, machine-processable form by means of concepts and their interrelations [4], and enabling the production and sharing of data that are commonly understood among human and software agents. Achieving the latter requires ensuring that the meaning of ontology elements is explicit and shareable, namely that all users have an unambiguous and consensual understanding of what each ontological element actually represents. In this paper we examine how **vagueness** affects shareability and reusability of semantic data. Vagueness is a common natural language phenomenon, demonstrated by concepts with blurred boundaries, like *tall*, *expert* etc., for which it is difficult to determine precisely their extensions (e.g. some people are borderline tall: neither clearly "*tall*" nor "*not tall*") [5].

Our position is threefold. i) That vagueness exists not only within isolated, application-specific, semantic data but also in public datasets that should be shareable and reusable. ii) That vagueness hampers the comprehensibility and shareability of these datasets and cause problems. iii) That the negative effects of vagueness can be partially tackled by making the data **vagueness-aware**, namely by annotating their elements with metainformation about the nature and characteristics of their vagueness. In the next section we explain and support the first two parts of our position with real world examples. In section 3 we describe how semantic data can become vagueness-aware via a vagueness metaontology. Sections 4 and 5 present related work and summarize our own.

2 Motivation and Approach Rationale

The possibility of vagueness in ontologies and semantic data has long been recognized in the research literature, especially in the area of Fuzzy Ontologies [3] [2]. An inspection of well-known ontologies and public semantic data reveals that the possibility is indeed a reality. A characteristic group of such elements are categorization relations where entities are assigned to categories with no clear applicability criteria. An example of such a relation is "*hasFilmGenre*", found in Linked Data datasets like LinkedMDB (`http://linkedmdb.org`) and DBpedia (`http://dbpedia.org`), that relates films with the genres they belong to. As most genres have no clear applicability criteria there will be films for which it is difficult to decide whether or not they belong to a given genre. A similar argument can be made for the DBpedia relations "*is dbpedia-owl:ideology of*" and "*dbpedia-owl:movement*". Another group of vague elements comprises specializations of concepts according to some vague property of them. Examples include "*Famous Person*" and "*Big Building*", in the Cyc Ontology (`http://www.cyc.com/platform/opencyc`), and "*Managerial Role*" and "*Competitor*", found in the Business Role Ontology (`http://www.ip-super.org`).

The presence of vague terms in semantic data often causes **disagreements** among the people who develop, maintain or use it. Such a situation arose in a real life scenario where we faced significant difficulties in defining concepts like "*Critical System Process*" or "*Strategic Market Participant*" while trying to develop an electricity market ontology. When, for example, we asked our domain experts to provide exemplary instances of critical processes, there was dispute among them about whether certain processes qualified. Not only did different domain experts have different criteria of process criticality, but neither could anyone really decide which of those criteria were sufficient for the classification. In other words, the problem was the vagueness of the predicate "*critical*". While disagreements may be overcome by consensus, they are inevitable as more users alter, extend, or use semantic data. A worse situation is when a user misinterprets the intended meaning of a vague term and uses it wrongly. Imagine an enterprise ontology where the concept "*Strategic Client*" was initially created and populated by the company's Financial Manager whose implicit criterion was the amount of revenue the clients generated for the company. Imagine also the new R&D Director querying the instances of this concept when crafting an R&D strategy. If their own applicability criteria for the term "*Strategic*" do not coincide with the Financial Manager's, using the returned list of clients might lead to poor decisions. The above examples show how the inherent context-dependence and subjectivity that characterizes vagueness may affect shareability in a negative way, due to potential disagreements or misunderstandings. More generally, typical use-case scenarios where this may happen include:

1. **Structuring Data with a Vague Ontology:** When domain experts are asked to define instances of vague concepts and relations, then disagreements may occur on whether particular entities constitute instances of them.
2. **Utilizing Vague Facts in Ontology-Based Systems:** When knowledge-based systems reason with vague facts, their output might not be optimal for those users who disagree with these facts.
3. **Integrating Vague Semantic Information:** When semantic data from several sources need to be merged then the merging of particular vague elements can lead to data that will not be valid for all its users.
4. **Evaluating Vague Semantic Datasets for Reuse:** When data practitioners need to decide whether a particular dataset is suitable for their needs, the existence of vague

elements can make this decision harder. It can be quite difficult for them to assess *a priori* whether the data related to these elements are valid for their application context.

To reduce the negative effects of vagueness, we put forward the notion of **vagueness-aware semantic data**, informally defined as "*semantic data whose vague ontological elements are accompanied by comprehensive metainformation that describes the nature and characteristics of their vagueness*". For example, a useful piece of metainformation is the set of applicability criteria that the element creator had in mind when defining the element (e.g. the amount of generated revenue as a criterion for a client to be strategic in the previous section's example). Another is the element creator itself (e.g. the author of a vague fact). In any case, our position is that having such metainformation, explicitly represented and published along with the vague semantic data, can improve the latter's comprehensibility and shareability, especially in regard to the four scenarios of the previous section. For example, the knowledge of the same vague concept's intended applicability criteria in two different datasets can i) prevent their merging in case these criteria are different and ii) help a data practitioner decide which of these two concepts's associated instances are more suitable for his/her application.

3 Making Ontologies Vagueness-Aware

3.1 Key Vagueness Aspects

In the literature two kinds of vagueness are identified: *quantitative-* or *degree-vagueness*; and *qualitative-* or *combinatory vagueness* [5]. A predicate has degree-vagueness if the existence of borderline cases stems from the lack of precise boundaries for the predicate along one or more dimensions (e.g. "*bald*" lacks sharp boundaries along the dimension of hair quantity while "*red*" can be vague for both brightness and saturation). A predicate has combinatory vagueness if there are a variety of conditions pertaining to the predicate, but it is not possible to make any crisp identification of those combinations which are sufficient for application. A classical example of this type is "*religion*" as there are certain features that all religions share (e.g. beliefs in supernatural beings, ritual acts) yet it is not clear which are able to classify something as a religion. Based on this typology, we suggest that for a given vague term it is important to represent and share the following explicitly:

- **The type of the term's vagueness**: Knowing whether a term has quantitative or qualitative vagueness is important as elements with an intended (but not explicitly stated) quantitative vagueness can be considered by others as having qualitative vagueness and vice versa.
- **The dimensions of the term's quantitative vagueness:** When the term has quantitative vagueness it is important to state explicitly its intended dimensions. E.g., if a CEO does not make explicit that for a client to be classified as strategic, its R&D budget should be the only pertinent factor, it will be rare for other company members to share the same view as the vagueness of the term "*strategic*" is multi-dimensional.

– **The necessary applicability conditions of the term's qualitative vagueness:** Even though a term with qualitative vagueness lacks a clear definition of sufficient conditions for objects to satisfy it, it can still be useful to define the conditions that are necessary for its applicability. This will not only narrow down the possible interpretations of the term (by including conditions that other people may forget or ignore) but will also provide better grounding on any discussion or debate that might arise about its meaning.

Furthermore, vagueness is **subjective** and **context dependent**. The first has to do with the same vague term being interpreted differently by different users. Two company executives might have different criteria for the term "*strategic client*". Even if they share an understanding of the type and dimensions of this term's vagueness, a certain amount of R&D budget (e.g. 1 million euros) makes a client strategic for one but not the other. Similarly, context dependence has to do with the same vague term being interpreted or applied differently in different contexts even by the same user; celebrating an anniversary is different to celebrating a birthday when it comes to judging how expensive a restaurant is. Therefore we additionally suggest that one should explicitly represent the term's **creator** as well as the **applicability context** for which it is defined or in which it is used.

3.2 A Metamodel of Vague Ontology Elements

Ontology elements that can be vague are typically concepts, relations, attributes and datatypes [2]. A concept is vague if – in the given domain, context or application scenario – it admits borderline cases; namely if there could be individuals for which it is indeterminate whether they instantiate the concept. Similarly, a relation is vague if there could be pairs of individuals for which it is indeterminate whether they stand in the relation. The same applies for attributes and pairs of individuals and literal values. Finally, a vague datatype consists of a set of vague terms which may be used within the ontology as attribute values (e.g. *performance* may take as values terms like *poor, mediocre* and *good*). To formally represent these vague elements by means of a metaontology, we consider the OWL metamodel defined in [6] and extend it by defining each vague element as a subclass of its corresponding element and by defining appropriate metaproperties that reflect the key aspects discussed in the previous sections. Figures 1 and 2 provide an overview of the metamodel while a concrete example of how this may be used to annotate a vague ontology is available at `http://boris.villazon.terrazas.name/data/VagueOntologyExample.ttl`

The metamodel is to be used by producers and consumers of semantic data, the former utilizing it to **annotate** the vague part of their ontologies with relevant metainformation and the latter **querying** this metainformation to better use them. Vagueness annotation is a manual task, meaning that knowledge engineers and domain experts should detect the vague elements, determine the relevant characteristics (type, dimensions, etc.) and populate the metamodel. How this task may be best facilitated is a subject for further research, but a good starting point would be the integration of the process within traditional semantic data production processes. Regarding the consumption of a vagueness-aware ontology, the first benefit it has for its potential users is that

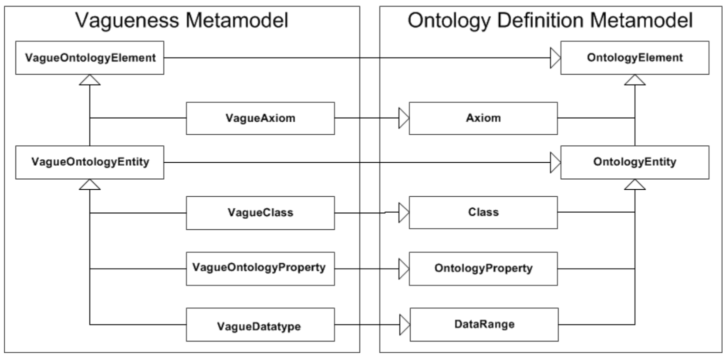


Fig. 1. Classes of Vagueness Metamodel

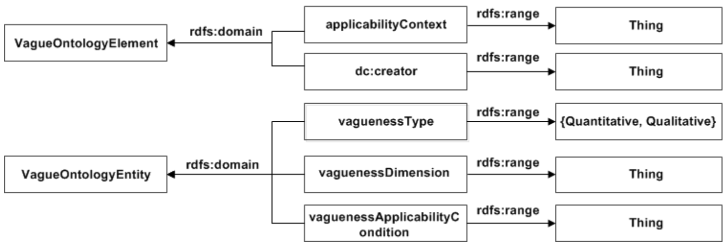


Fig. 2. Properties of Vague Elements

it makes them aware of the existence of vagueness in the domain. This is important because vagueness is not always obvious, meaning it can easily be overlooked and cause problems. The second benefit is that the ontology's users may query each of the vague elements' metainformation and use it in order to reduce these problems.

For example, when structuring data with a vague ontology, disagreements may occur on whether particular objects are instances of vague concepts. If, however, information like the applicability conditions and contexts of these elements are known to the people who perform this task, then their possible interpretation spaces will be reduced. Also, when vague elements are used within some end-user application, the availability of vagueness metainformation can help the system's developers in two ways. i) It will make them aware of the fact that the ontology contains vague information and thus some of the system's output might not be considered accurate by the end-users. ii) They may use the vagueness metainformation to try to deal with that. For example, the applicability context of a vague axiom can be used in a recommendation system to explain why a particular item was recommended. Finally, in dataset integration and evaluation scenarios, the vagueness metamodel can be used to compare ontologies' vagueness compatibility. For example, if the same two vague classes have different vagueness dimensions, then the one class's set of instance membership axioms might not be appropriate for the second's as it may have been defined with a different vagueness interpretation in mind. A simple query to the two ontologies' vagueness metamodel could reveal this issue.

4 Related Work

Representing semantic data metainformation is common in the community, like the VoID vocabulary for describing Linked datasets [1]. However, no vagueness-related vocabularies are yet available. In a more relevant approach an OWL 2 model for representing fuzzy ontologies is defined [3]. It focuses, however, on enabling the representation of fuzzy degrees and fuzzy membership functions within an ontology, without any information regarding the intended meaning of the fuzzy elements' vagueness or the interpretation of their degrees (e.g. the dimensions a concept membership degree covers). Thus, our approach is complementary to fuzzy ontology related works, in the sense that it may be used to enhance the comprehensibility of fuzzy degrees.

5 Conclusions and Future Work

In this paper we considered vagueness in semantic data and we demonstrated the need and potential benefits of making the latter vagueness-aware by annotating their elements with a metaontology that explicitly describes the vagueness's nature and characteristics. The idea is that even though the availability of the metainformation will not eliminate vagueness, it will manage to reduce the high level of disagreement and low level of comprehensibility it may cause. This increased semantic data comprehensibility and shareability we intend to establish in our future work through user-based experiments.

Acknowledgement

The research has been funded from the People Programme (Marie Curie Actions) of the European Union's 7th Framework Programme P7/2007-2013 under REA grant agreement n^o 286348.

References

1. Alexander, K., Cyganiak, R., Hausenblas, M., Zhao, J.: Describing Linked Datasets On the Design and Usage of VoID , the "Vocabulary Of Interlinked Datasets", VoID working group, 2009.
2. Alexopoulos, P., Wallace, M., Kafentzis, K., Askounis, D.: IKARUS-Onto: A Methodology to Develop Fuzzy Ontologies from Crisp Ones. Knowledge and Information Systems, 32(3):667-695, September 2012.
3. Bobillo, F., Straccia, U.: Fuzzy ontology representation using OWL 2. International Journal of Approximate Reasoning, 52(7):1073-1094, October 2011.
4. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: What are ontologies, and why do we need them. IEEE Intelligent Systems, pp 20-26, 1999.
5. Hyde, D.: Vagueness, Logic and Ontology. Ashgate New Critical Thinking in Philosophy, 2008.
6. Vrandecic, D., Volker, J., Haase, P., Tran, D.T., Cimiano, P.: A Metamodel for Annotations of Ontology Elements. In Proceedings of the 2nd Workshop on Ontologies and Meta-Modeling, 2006.

A GUI for MLN

Estêvão F. Aguiar[1], Marcelo Ladeira[1], Rommel N. Carvalho[1], and Shou Matsumoto[1]

Department of Computer Science
University of Brasília
Campus Universitario Darcy Ribeiro
Brasília, Distrito Federal, Brazil
`estevaofaguiar@gmail.com`, `mladeira@unb.br`,
`{rommel.carvalho,cardialfly}@gmail.com`
`http://www.cic.unb.br`

Abstract. This paper focuses on the incorporation of the Markov Logic Network (MLN) formalism as a plug-in for UnBBayes, a Java framework for probabilistic reasoning based on graphical models. MLN is a formalism for probabilistic reasoning which combines the capacity of dealing with uncertainty tolerating imperfections and contradictory knowledge based a Markov Network (MN) with the expressiveness of First Order Logic. A MLN provides a compact language for specifying very large MNs and the ability to incorporate, in modular form, large domain of knowledge (expressed in First Order Logic sentences) inside itself. A Graphical User Interface for the software Tuffy was implemented into UnBBayes to facilitate the creation, and inference of MLN models. Tuffy is a Java open source MLN engine.

Keywords: Markov Logic Network, MLN, Tuffy, UnBBayes, Markov Network, probabilistic reasoning, probabilistic graphical models

1 Introduction

In the past decade, several languages have been proposed to deal with complex knowledge representation problems that also need to deal with uncertainty. A frequent approach is to combine both logic and probabilistic formalisms resulting in a powerful model for knowledge representation and treatment of uncertainty. Some examples of these approaches were build and have been improved every day as Markov Logic Networks (MLN) [1], Multi-Entity Bayesian Networks (MEBN) [3], Probabilistic Relational Models (PRM) [2], Relational Markov Networks (RMN) [10], and Structural Logistic Regression (SLR) [7].

Markov Logic Network (MLN) is a principled formalism which combines First-Order Logic (FOL) with Markov network (MN). An MLN, basically, is a first-order knowledge base where a weight is assigned to each formula. The weight of a formula indicates how strong the formula is as a constraint. Together with a finite set of constants, an MLN can be grounded as a Markov network. This way, a MLN can be seen as a template for building Markov networks [1].

There are a few implementations for MLN like Alchemy [1] in C++, Tuffy [6] in Java, ProbCog [11] in both Python and Java, and Markov TheBeast [9] in Java. In some of them there is no graphical user interface (GUI). The one that does, the interface is quite simple providing no real ease-of-use. In general, as these software are not very friendly, they become hard to use for users without previous experience with their programming tasks and command lines.

This paper presents an implementation of a Java tool that consists of a GUI to facilitate the task of making inferences, creating, and editing MLN models. This tool was developed at the University of Brasilia (UnB) and uses the software Tuffy as a library. Its current features include GUIs for modeling terms of a knowledge base into a tree structure and for searching them in order to help the user find terms easily in large models. Moreover, it is also possible to edit and to persist these structures as a standard MLN file (compatible with both Tuffy and Alchemy). Besides that, every parameter that can be set on Tuffy can be easily set in the GUI. It even supports the addition in the GUI of new parameters that might be present in future versions of Tuffy using only a configuration file. This tool was implemented as a plug-in for UnBBayes, a Java open source software developed at UnB that is capable of modeling, making inferences and learning probabilistic networks [4].

This paper is structured as follows. Section 2 presents the MLN. Section 3 overviews some implementations of MLN and presents the major reasons for choosing Tuffy as the application programming interface (API) behind this plug-in. Section 4 introduces the GUI developed as a plug-in for UnBBayes.

2 Markov Logic Networks

A knowledge base of First-Order Logic (FOL) can be viewed as a set of constraints on possible worlds. Each formula has an associated weight that reflects how strong this formula is as a constraint [1]. An MLN is a set with formulas of FOL assigned to a real-valued weight for each formula. Together with a finite set of constants, it defines a Markov Network (MN). Each state of the MN generated represents a possible world of the generic MLN representation. A possible world determines the truth value of each ground (*i.e.* instantiated) predicate. Thus, it is said that an MLN is like a template for constructing MNs. Given different set of constants, it will produce different MNs with different values and sizes. However they have the same parameters and regularities in structure. Different instantiated formulas still have the same weights. So, in MLN it is possible to use inference methods generally used for MNs, since the used network is a grounded one. However, due to the fact that most of time the grounded network is large and complex, to use this method could be infeasible. Therefore, approximate and lifted inference algorithms have been proposed [1].

Maximum a Posteriori (MAP) inference (*i.e.* finding the most likely state of the world consistent with some evidence) and marginal inference (*i.e.* computing arbitrary conditional probabilities) are common approaches to making inferences in MLN. Learning algorithms are used to build, from historic data, models that represent a problem to be treated. For this formalism, learning methods are used to construct or refine a MLN. Two types of learning are specified: weight learning (*i.e.* which tries to find the weights of the specified formulas that maximize the likelihood or conditional likelihood of a relational database) and the harder technique of structure learning (*i.e.* which tries to learn the formulas themselves).

More details on MLN can be found in [1] and will not be covered in this paper.

3 The choice of an implementation

With the intent of building a GUI for MLN, the first step is to implement or find an existing implementation of the formalism. So, pros and cons of some implementations have to be analyzed. If no implementation had compatibility with UnBBayes, it would be necessary to create a new one. Fortunately it was not the case. The pros and cons of the more common implementations are presented below. As our goal was to build a plug-in for UnBBayes, the programming language had a larger weight than the features available on the tool. UnBBayes [4] is an open source application developed in Java that provides a framework for building and reasoning with probabilistic graphical models. Since version 1.5.1, it works with Java Plugin Framework (JPF). JPF allows the construction of scalable projects, loading plug-ins at runtime. The MLN GUI has been built as a plug-in for UnBBayes.The software analyzed were Alchemy [1], ProbCog [11], TheBeast [9] and Tuffy [6].

3.1 Alchemy

Alchemy is the reference for other implementations of MLN and is the most complete of them. It covers MAP Inference, marginal inference, weight learning, structure learning and other features from each of the mentioned topics. Alchemy is an open source software developed in C/C++. It does not have a GUI and it works only in Linux or Linux shell emulator. Alchemy was the first option to extend, but its programing language is not easily integrated with Java.

3.2 TheBeast

TheBeast [8] is an open source and is a Statistical Relational Learning software based on MLN. Although it is developed in Java, it does not have much documentation and it does not work similarly to Alchemy. This fact impacts on that it would be harder to work with it. TheBeast has no GUI implemented either.

3.3 ProbCog

ProbCog is an open source software for Statistical Relational Learning that supports learning and inference for relational domains. Merged to ProbCog, is PyMLN, a toolbox and a software library for learning and reasoning in MLN.

It has a GUI for MLN but it, seemingly, shows the necessary files for inference and the main parameters to be more easily selected, but nothing beyond the basic. Most of the code of ProgCog is developed in Java, although its MLN tool is developed in Python.

3.4 Tuffy

Tuffy is an open source Markov Logic Network engine. It is developed in Java and makes use of a PostgreSQL database. Tuffy is in version 0.3 and is capable of MRF partitioning, MAP inference, marginal inference and weight learning. Since Tuffy has many similar features to Alchemy, as weel as the same structure for input files, it has no GUI, and it is implemented in the same programming language as UnBBayes, it ended up being the most suitable MLN implementation to be used in the MLN GUI plug-in.

4 The GUI for MLN

There are several helpful easy to use GUI tools for Bayesian networks. However, this is not true to MLN yet. For most of them, the only way to make it work is to set command line parameters and then enter commands through a console. Sometimes you must memorize a bunch of commands if you want to realize a task fast, while you could just press buttons and choose options with some clicks in a more easy to use GUI interface. Creating a GUI to simplify this process of designing and using MLNs was the main motivation of this research. The following paragraphs describe the main features of a proposed GUI for MLN.

This project of a GUI for MLN into UnBBayes was built as a JPF plug-in. The plug-in structure provides a way to run a new project inside the running environment of UnBBayes. The bind between the new plug-in and the core of UnBBayes happens in a way that no changes are needed in the core structure.

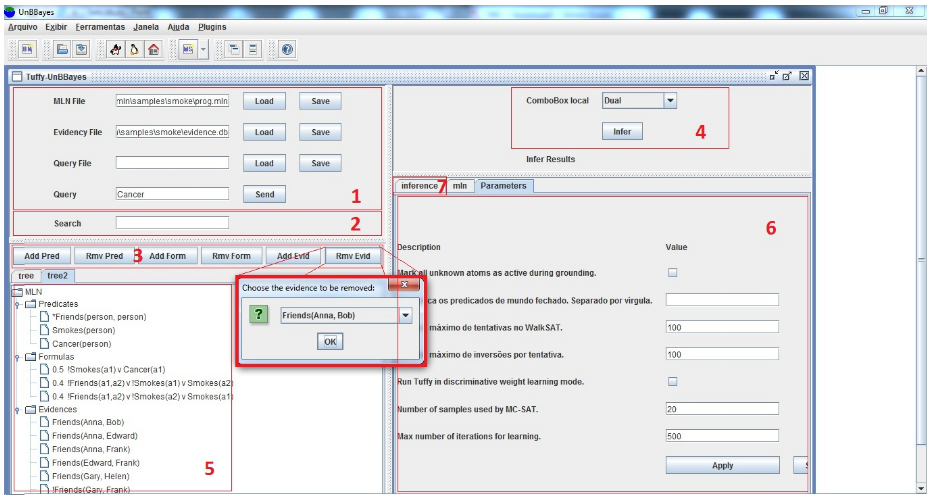
Basically, building new plug-in implementations for UnBBayes is really simple, since a stub implementation is available in [5].

Figure 1 presents the GUI divided in numbered parts. Each part is described bellow.

The Tuffy input files are: a MLN file (.mln), an evidence file (.db) and a query file (.db). The last one can be replaced passing its content through command line. Figure 1 Part 1 shows the possibility to load this three files and the possibility to send the query predicates through a text field.

When the MLN file and the evidence file are loaded, their terms (*i.e.* predicates, weighted formulas and evidences) are separated and organized in a tree structure as shown in Figure 1 Part 5. This tree structure gives a great gain of visualization and differences between structures into the MLN.

Figure 1 Part 2 presents a very useful search tool. It searches dynamically predicates, formulas, and evidence that match the inputted string. This feature is useful when working with very large MLNs.



MLN File: mln\samples\smoke\prog.mln
Evidency File: i\samples\smoke\evidence.db
Query File:
Query: Cancer
Search:

MLN
- Predicates
  - *Friends(person, person)
  - Smokes(person)
  - Cancer(person)
- Formulas
  - 0.5 !Smokes(a1) v Cancer(a1)
  - 0.4 !Friends(a1,a2) v !Smokes(a1) v Smokes(a2)
  - 0.4 !Friends(a1,a2) v !Smokes(a2) v Smokes(a1)
- Evidences
  - Friends(Anna, Bob)
  - Friends(Anna, Edward)
  - Friends(Anna, Frank)
  - Friends(Edward, Frank)
  - Friends(Gary, Helen)
  - !Friends(Gary, Frank)

Choose the evidence to be removed: Friends(Anna, Bob)

ComboBox local: Dual

Description	Value
Mark all unknown atoms as active during grounding.	☐
ca os predicados de mundo fechado. Separado por vírgula.	
máximo de tentativas no WalkSAT.	100
máximo de inversões por tentativa.	100
Run Tuffy in discriminative weight learning mode.	☐
Number of samples used by MC-SAT.	20
Max number of iterations for learning.	500

Fig. 1. GUI for MLN implemented as plugin into UnBBayes

The GUI also presents a way to add and remove predicates, formulas and evidence. This feature is shown in Figure 1 Part 3. Lots of terms can be directly inputted into the correct classification. The deletion is made from a drop down list which brings to the user all the existing terms. Every change made through this feature is persisted in the original file. This feature makes it easier for the user to include or remove terms in a MLN model.

Figure 1 Part 4 allows the user to choose what inference method to use and the button to trigger the inference process, which will be executed by Tuffy in the background. Tuffy is embedded into UnBBayes and used as a library through its API.

Figure 1 Part 7 is displayed when the "inference" tab is chosen. It presents the output in a text area, the same way that it is presented in the output file in Tuffy.

Figure 1 Part 6 presents the parameters of Tuffy in an easy way to set and save. The parameters of Tuffy were parameterized by type that they represent (*e.g.* integer, float, boolean and string). This allows the parameters to be loaded to the interface from a configuration file and new parameters added in new versions of Tuffy can be easily incorporated to UnBBayes without the need to change any programming code. The dynamic values of the parameters are defined in another configuration file.

5 Conclusion

This paper presents a GUI for Tuffy, a Java Markov Logic Network inference engine. As shown, this GUI facilitates the task of creating MLNs models and reasoning with them. This GUI was implemented as a JFP plug-in for the

UnBBayes software. UnBBayes and this plug-in[1] is available from `http://unbbayes.sourceforge.net/` under GPL license.

References

1. P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for AI.* Morgan and Claypool, 2009.
2. Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *International Joint Conference on Artificial Intelligence*, volume 16, pages 1300–1309. LAWRENCE ERLBAUM ASSOCIATES LTD, 1999.
3. K.B. Laskey. MEBN: A Language for First-Order Bayesian Knowledge Bases. *Artificial Intelligence, 172(2-3): 140-178*, 2008.
4. M. Vieira M. Onishi R.N. Carvalho M. Ladeira, D. da Silva and W. da Silva. Platform independent and open tool for probabilistic networks. *Proceedings of the IV Artificial Intelligence National Meeting (ENIA 2003) on the XXIII Congress of the Brazilian Computer Society (SBC 2003), Unicamp, Campinas, Brazil*, 2003.
5. 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.
6. R C. Doan A. Shavlik J. Niu, F. Tuffy: Scaling up statistical inference in markov logic networks using an RDBMS. *Proceedings of the VLDB Endowment, 4(6), 373-384*, 2011.
7. Alexandrin Popescul and Lyle H Ungar. Structural logistic regression for link analysis. *Departmental Papers (CIS)*, page 133, 2003.
8. Sebastian Riedel. Improving the accuracy and efficiency of MAP inference for Markov Logic. In *Proceedings of the 24th Annual Conference on Uncertainty in AI (UAI '08)*, pages 468–475, 2008.
9. Sebastian Riedel. Markov thebeast user manual. 2008.
10. Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 485–492. Morgan Kaufmann Publishers Inc., 2002.
11. Jain D. Beetz M. Tenorth, M. Knowledge representation for cognitive robots. *Knstliche Intelligenz, Springer, volume 24*, 2010.

[1] This plug-in can only be downloaded from the SVN repository. Soon a distribution will be released for simple download and installation.