Operations over Lightweight Ontologies

Marco A. Casanova[1], José A.F. de Macêdo[2], Eveline R. Sacramento[1,3], Ângela M.A. Pinheiro[2], Vânia M.P. Vidal[2], Karin K. Breitman[1], and Antonio L. Furtado[1]

[1] Department of Informatics – PUC-Rio – Rio de Janeiro, RJ – Brazil
{casanova,karin,furtado,esacramento}@inf.puc-rio.br
[2] Department of Computing, Federal University of Ceará – Fortaleza, CE – Brazil
{jose.macedo,angelap,vvidal,}@lia.ufc.br
[3] Ceará State Foundation of Meteorology and Water Resources – Fortaleza, CE – Brazil
eveline@funceme.br

Abstract. The best practices for Linked Data design recommend reusing known ontologies. However, the process of reusing an ontology involves two issues: (1) selecting a set of terms from the ontology vocabulary; and (2) using the ontology constraints to derive those that apply to such terms. The first issue is quite simple and corresponds to the familiar practice of importing namespaces. This paper proposes to address the second issue by introducing a set of operations that create new ontologies, including their constraints, out of other ontologies. The operations treat ontologies as theories and not just as vocabularies. The paper proceeds to show how to compute the operations for *lightweight ontologies*, that is, ontologies built upon DL-Lite core with arbitrary number restrictions. It also addresses the question of minimizing the set of constraints of a lightweight ontology that results from an operation. Finally, the paper describes a tool that implements the operations and offers other facilities to manipulate lightweight ontologies.

Keywords: constraints, Linked Data, DL-Lite core, Description Logics.

1 Introduction

The term *Linked Data* refers to a set of best practices for publishing and connecting structured data on the Web [5]. The 'Linked Data Principles' [4] provide a basic recipe for publishing and connecting data using the infrastructure of the Web. From an application development perspective, Linked Data has the following characteristics [6]:

1. Data is strictly separated from formatting and presentational aspects.
2. Data is self-describing. If an application consuming Linked Data encounters data described with an unfamiliar vocabulary, the application can dereference the URIs that identify vocabulary terms in order to find their definition.
3. The use of HTTP as a standardized data access mechanism and RDF as a standardized data model simplifies data access compared to Web APIs, which rely on heterogeneous data models and access interfaces.
4. The Web of Data is open, meaning that applications do not have to be implemented against a fixed set of data sources, but they can discover new data sources at run-time by following RDF links.

R. Meersman et al. (Eds.): OTM 2012, Part II, LNCS 7566, pp. 646–663, 2012.

We are particularly interested in the second characteristic. The definition of vocabulary terms must ultimately include a set of constraints that capture the semantics of the terms. Therefore, when publishing Linked Data, we argue that the designer should go further and analyze the constraints of the ontologies from which he is drawing the terms to construct his vocabulary.

In this paper, we first introduce a set of operations on ontologies that create new ontologies, including their constraints, out of other ontologies. Such operations extend the idea of namespaces to take into account constraints and treat ontologies as theories, i.e., sets of constraints over a given vocabulary. Then, we concretely show how to compute the operations when the ontologies are built upon DL-Lite core with arbitrary number restrictions [2]. We refer to such ontologies as *lightweight ontologies*. We also discuss how to minimize the set of constraints of a lightweight ontology that results from an operation. Finally, we describe a tool that implements the operations and offers other facilities to manipulate lightweight ontologies.

Applications may benefit from this design strategy as follows. Consider a Linked Data source $\boldsymbol{S}$ whose data is published according to an ontology O_0, based on known ontologies $O_1,\ldots,O_n$. Assume that the constraints of O_0 are derived as argued in the preceding paragraphs and that the published data is consistent with such constraints. Then, in general, any application that accesses $\boldsymbol{S}$ and that is prepared to process data consistent with $O_1,\ldots,O_n$ will also be able to process data published by $\boldsymbol{S}$, since the application expects data consistent with the constraints derived from those of $O_1,\ldots,O_n$ that apply to the classes and properties in the vocabulary of O_0. For example, the application may be a domain-specific data mashup tool that has a built-in optimizer that knows how to take advantage of the constraints of $O_1,\ldots,O_n$ to optimize the construction of the mashup.

We adopt the machinery to handle constraints developed in [8][9][10], which uses DL-Lite core with arbitrary number restrictions [2]. Previous work by the authors [9] introduced the notion of *open fragment*, which is captured by the projection operation. However, none of the previous work by the authors considered the other operations, the question of optimizing the representation of the resulting constraints or described a tool to implement the operations.

The paper is organized as follows. Section 2 presents the formal definition of the operations. Section 3 shows how to compute the operations for lightweight ontologies. Section 4 describes the *OntologyManagement tool* that implements the operations and offers other facilities to manipulate lightweight ontologies. Section 5 summarizes related work. Finally, Section 6 contains the conclusions.

2 A Formal Framework

2.1 A Brief Review of Basic Concepts

The definition of the operations depends only on the notion of theory, which we introduce in the context of Description Logic (DL) [3].

A DL *language* $\mathcal{L}$ is characterized by a *vocabulary V*, consisting of a set of *atomic concepts*, a set of *atomic roles*, and the *bottom concept* $\bot$. The sets of *concept descriptions* and *role descriptions* of *V* depend on the specific description logic. An *inclusion* of *V* is an expression of the form $u \sqsubseteq v$, where *u* and *v* both are concept descriptions or both are role descriptions.

An *interpretation* $\boldsymbol{s}$ of *V* consists of a nonempty set Δ^s, the *domain* of $\boldsymbol{s}$, and an *interpretation function*, also denoted $\boldsymbol{s}$, with the usual definition [3]. We use $\boldsymbol{s}(u)$ to indicate the value that $\boldsymbol{s}$ assigns to an expression *u* of *V*.

Let σ be an inclusion of *V* and Σ be a set of inclusions of *V*. We say that: $\boldsymbol{s}$ *satisfies* σ or $\boldsymbol{s}$ is a *model* of σ, denoted $\boldsymbol{s} \models \sigma$, iff $\boldsymbol{s}(u) \subseteq \boldsymbol{s}(v)$; $\boldsymbol{s}$ *satisfies* Σ or $\boldsymbol{s}$ is a *model* of Σ, denoted $\boldsymbol{s} \models \Sigma$, iff $\boldsymbol{s}$ satisfies all inclusions in Σ; Σ *logically implies* σ, denoted $\Sigma \models \sigma$, iff any model of Σ satisfies σ; Σ is *satisfiable* or *consistent* iff there is a model of Σ; Σ is *strongly consistent* iff Σ is *consistent* and Σ does not logically imply $A \sqsubseteq \bot$, for some atomic concept *A*.

Let Σ be a set of inclusions of *V*. The *theory* of Σ in *V*, denoted $\tau[\Sigma]$ is the set of all inclusions of *V* which are logical consequences of Σ.

We are specially interested in *DL-Lite core with arbitrary number restrictions* [2], denoted $DL\text{-}Lite_{core}^{\mathcal{N}}$. The sets of *basic concept descriptions*, *concept descriptions* and *role descriptions* in this case are defined as follows:

- If *P* is an atomic role, then *P* and P^- (*inverse role*) are role descriptions
- If *u* is an atomic concept or the bottom concept, and *p* is a role description, then *u* and $(\geq n\ p)$ (*at-least restriction*, where *n* is a positive integer) are basic concept descriptions and also concept descriptions
- If *u* is a concept description, then $\neg u$ (*negated concept*) *is* a concept description

We use the following abbreviations: "$\top$" for "$\neg\bot$" (*universal concept*), "$\exists p$" for "$(\geq 1\ p)$" (*existential quantification*), "$(\leq n\ p)$" for "$\neg(\geq n+1\ p)$" (*at-most restriction*) and "$u \mid v$" for "$u \sqsubseteq \neg v$" (*disjunction*). By an *unabbreviated* expression we mean an expression that does not use such abbreviations.

For $DL\text{-}Lite_{core}^{\mathcal{N}}$, an *inclusion* of *V* is an expression of one of the forms $u \sqsubseteq v$ or $u \sqsubseteq \neg v$, where *u* and *v* both are basic concept descriptions (including the bottom concept $\bot$). That is, an inclusion may contain a negated concept only on the right-hand side and it may not involve role descriptions.

Finally, an *ontology* is a pair $\boldsymbol{O}=(V,\Sigma)$ such that *V* is a finite vocabulary, whose atomic concepts and atomic roles are called the *classes* and *properties* of $\boldsymbol{O}$, respectively, and Σ is a finite set of inclusions of *V*, called the *constraints* of $\boldsymbol{O}$. A *lightweight ontology* is an ontology whose constraints are inclusions of $DL\text{-}Lite_{core}^{\mathcal{N}}$. From this point on, we will use the terms class, property and constraint instead of atomic concept, atomic role and inclusion, whenever reasonable.

Table 1 lists the types of $DL\text{-}Lite_{core}^{\mathcal{N}}$ inclusions that represent constraints commonly used in conceptual modeling.

Table 1. Common constraint types used in conceptual modeling

Constraint Type	Abbreviated form	Unabbreviated form	Informal semantics
Domain Constraint	$\exists P \sqsubseteq C$	$(\geq 1\ P) \sqsubseteq C$	property P has class C as domain, that is, if (a,b) is a pair in P, then a is an individual in C
Range Constraint	$\exists P^- \sqsubseteq C$	$(\geq 1\ P^-) \sqsubseteq C$	property P has class C as range, that is, if (a,b) is a pair in P, then b is an individual in C
minCardinality Constraint	$C \sqsubseteq (\geq k\ P)$ or $C \sqsubseteq (\geq k\ P^-)$	$C \sqsubseteq (\geq k\ P)$ or $C \sqsubseteq (\geq k\ P^-)$	property P or its inverse P^- maps each individual in class C to at least k distinct individuals
maxCardinality Constraint	$C \sqsubseteq (\leq k\ P)$ or $C \sqsubseteq (\leq k\ P^-)$	$C \sqsubseteq \neg(\geq k+1\ P)$ or $C \sqsubseteq \neg(\geq k+1\ P^-)$	property P or its inverse P^- maps each individual in class C to at most k distinct individuals
Subset Constraint	$C \sqsubseteq D$	$C \sqsubseteq D$	each individual in C is also in D, that is, class C denotes a subset of class D
Disjointness Constraint	$C \mid D$	$C \sqsubseteq \neg D$	no individual is in both C and D, that is, classes C and D are disjoint

2.2 Definition of the Operations

We define the following operations over ontologies.

Definition 1: Let $O_1 = (V_1, \Sigma_1)$ and $O_2 = (V_2, \Sigma_2)$ be two ontologies, W be a subset of V_1 and Φ be a set of constraints over V_1.

(i) The *selection* of $O_1 = (V_1, \Sigma_1)$ for Φ, denoted $\sigma[\Phi](O_1)$, returns the ontology $O_S = (V_S, \Sigma_S)$, where $V_S = V_1$ and $\Sigma_S = \Sigma_1 \cup \Phi$.

(ii) The *projection* of $O_1 = (V_1, \Sigma_1)$ over W, denoted $\pi[W](O_1)$, returns the ontology $O_P = (V_P, \Sigma_P)$, where $V_P = W$ and Σ_P is the set of constraints in $\tau[\Sigma_1]$ that use only symbols in W.

(iii) The *union* of $O_1 = (V_1, \Sigma_1)$ and $O_2 = (V_2, \Sigma_2)$, denoted $O_1 \cup O_2$, returns the ontology $O_U = (V_U, \Sigma_U)$, where $V_U = V_1 \cup V_2$ and $\Sigma_U = \Sigma_1 \cup \Sigma_2$.

(iv) The *intersection* of $O_1 = (V_1, \Sigma_1)$ and $O_2 = (V_2, \Sigma_2)$, denoted $O_1 \cap O_2$, returns the ontology $O_N = (V_N, \Sigma_N)$, where $V_N = V_1 \cap V_2$ and $\Sigma_N = \tau[\Sigma_1] \cap \tau[\Sigma_2]$.

(v) The *difference* of $O_1 = (V_1, \Sigma_1)$ and $O_2 = (V_2, \Sigma_2)$, denoted $O_1 - O_2$, returns the ontology $O_D = (V_D, \Sigma_D)$, where $V_D = V_1$ and $\Sigma_D = \tau[\Sigma_1] - \tau[\Sigma_2]$.

Note that selections can be defined as unions (see the example at the end of Section 2.3). We also observe that the ontology that results from each operation is unique. However, the set of constraints of the resulting ontology is not necessarily minimal, a point elaborated in Section 3.3.

Recall that, in [9], we introduced the notions of open and closed fragments of an ontology. The operations capture these notions as follows. Let $O_1 = (V_1, \Sigma_1)$ be an ontology and W be a subset of V_1. The *open fragment* of O_1 *defined by* W is the projection $\pi[W](O_1)$ of O_1 over W, and the *closed fragment* of O_1 *defined by* W is captured by $\sigma[\Phi](\pi[W](O_1))$, where Φ contains an inclusion of the form $A \sqsubseteq \bot$, for each atomic concept A in V_1 but not in W, and an inclusion of the form $(\geq 1\ P) \sqsubseteq \bot$, for each atomic role P in V_1 but not in W (these inclusions force the interpretations of A and P to be the empty set).

2.3 Examples

As outlined in the introduction, when publishing Linked Data, we argue that the designer should analyze the constraints of the ontologies he uses to construct his vocabulary. We further motivate this argument with the help of the *Music Ontology* [19].

The *Music Ontology* (MO) provides concepts and properties to describe artists, albums, tracks, performances, arrangements, etc. on the Semantic Web. It is used by several Linked Data sources, including MusicBrainz and BBC Music. The *Music Ontology* RDF schema uses terms from the *Friend of a Friend* (FOAF) [7] and the XML Schema (XSD) vocabularies. We respectively adopt the prefixes "mo:", "foaf:" and "xsd:" to refer to these vocabularies.

Figure 1 shows the class hierarchies of MO rooted at classes foaf:Agent and foaf:Person. Let us focus on this fragment of MO.

We first recall that FOAF has a constraint, informally formulated as follows:

- foaf:Person and foaf:Organization are disjoint classes

Let V_1 be the following set of terms from the FOAF and the XSD vocabularies, and V_2 contains the rest of the terms that appear in Figure 1:

V_1={ foaf:Agent, foaf:Person, foaf:Group, foaf:Organization, foaf:name, xsd:string }
V_2={ mo:MusicArtist, mo:CorporateBody, mo:SoloMusicArtist, mo:MusicGroup, mo:Label, mo:member_of }

Let $\boldsymbol{O_1}=(V_1,\Sigma_1)$ be the ontology obtained by the *projection* of FOAF over V_1, denoted $\pi[V_1](FOAF)$ and defined in such a way that Σ_1 is the set of constraints over V_1 that are logical consequences of the constrains of FOAF. We stress that Σ_1 captures the semantics of the terms in V_1 defined in FOAF and contains the following constraints:

Σ_1={ (≥1 foaf:name) ⊑ foaf:Person, (≥1 foaf:name$^-$) ⊑ xsd:string,
foaf:Person ⊑ ¬foaf:Organization, foaf:Group ⊑ foaf:Agent,
foaf:Organization ⊑ foaf:Agent }

Let $\boldsymbol{O_2}=(V_2,\Sigma_2)$ be such that Σ_2 contains just the subset constraints over V_2 shown in Figure 1:

Σ_2={ mo:SoloMusicArtist ⊑ mo:MusicArtist, mo:MusicGroup ⊑ mo:MusicArtist,
mo:Label ⊑ mo:CorporateBody }

Then, most of Figure 1 is captured by the *union* of $\boldsymbol{O_1}$ and $\boldsymbol{O_2}$, defined as the ontology $\boldsymbol{O_3}=(V_3,\Sigma_3)$, where $V_3 = V_1 \cup V_2$ and $\Sigma_3 = \Sigma_1 \cup \Sigma_2$.

The terms and constraints shown in Figure 1, but not included in $\boldsymbol{O_3}$, are obtained by the *selection* of $\boldsymbol{O_3}$ using the set of constraints Σ_5, where

Σ_5={ mo:SoloMusicArtist ⊑ foaf:Person, mo:MusicGroup ⊑ foaf:Group,
mo:CorporateBody ⊑ foaf:Organization,
∃mo:member_of ⊑ foaf:Person, ∃mo:member_of$^-$ ⊑ foaf:Group }

The selection, denoted $\sigma[\Sigma_5](\boldsymbol{O_3})$, returns the ontology $\boldsymbol{O_4}=(V_4,\Sigma_4)$, where $V_4 = V_3$ and $\Sigma_4 = \Sigma_3 \cup \Sigma_5$. Note that the selection operation is just a convenience since we may define $\boldsymbol{O_4}=(V_4,\Sigma_4)$ as the union of $\boldsymbol{O_3}=(V_3,\Sigma_3)$ with a new ontology $\boldsymbol{O_5}=(V_3,\Sigma_5)$ (the ontologies have the same vocabulary).

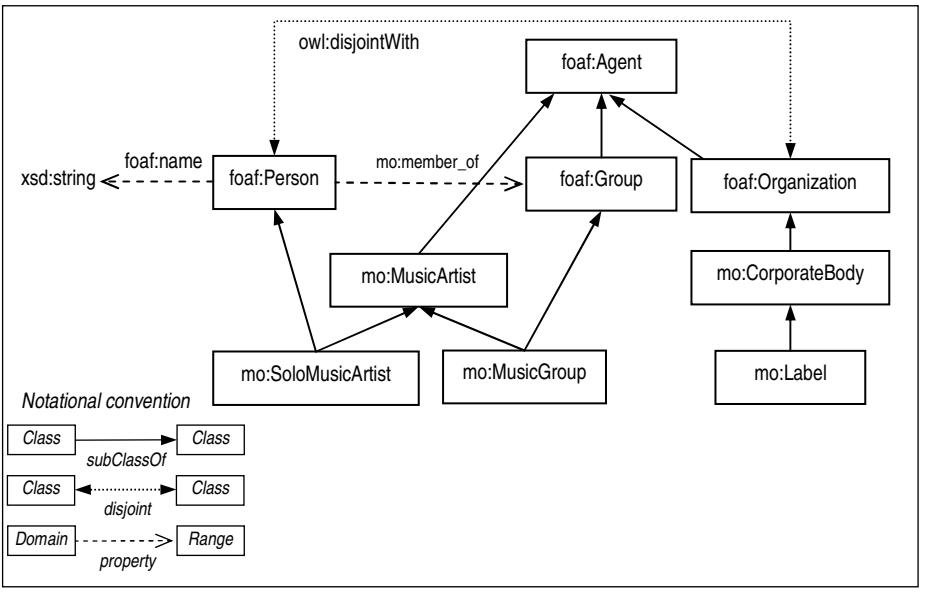

Fig. 1. The class hierarchies of *MO* rooted at classes foaf:Agent and foaf:Person

Finally, we construct $\boldsymbol{O_0}=(V_0,\Sigma_0)$, the ontology that corresponds to Figure 1, in two different, albeit equivalent ways:

(1) $\boldsymbol{O_0} = \sigma[\Sigma_5](\pi[V_1](FOAF) \cup \boldsymbol{O_2})$, using the selection operation

(2) $\boldsymbol{O_0} = ((\pi[V_1](FOAF) \cup \boldsymbol{O_2}) \cup \boldsymbol{O_5})$, eliminating the selection operator

The reader is invited to reflect upon the definition of $\boldsymbol{O_0}$ is Eq. (1) (or in Eq. (2)) above. We contend that the expression of the right-hand side of Eq. (1) provides a quite reasonable explanation of how $\boldsymbol{O_0}$ is constructed from FOAF and additional terms and constraints.

As a second simple example, consider the problem of designing a Linked Data source about classical music. One might start by projecting the Music Ontology on the concepts and properties of interest, including mo:Genre (not shown in Figure 1), a class of the Music Ontology that holds any taxonomy of musical genres, and mo:hasGenre, an object property with domain mo:MusicArtist and range mo:Genre. To define a selection of this fragment of the Music Ontology that refers to classical music, it suffices to add a new constraint saying that mo:Genre must be a subset of the classical music genres (defined by enumeration, say).

3 Computing the Operations over Lightweight Ontologies

In this section, we show how to efficiently compute the operations over lightweight ontologies, using the notion of constraint graphs.

3.1 Constraint Graphs

We say that the *complement* of a basic concept description e is $\neg e$, and vice-versa. If c is a concept description, then $\overline{c}$ denotes the complement of c.

Let Σ be a set of (unabbreviated) inclusions and Ω be a set of (unabbreviated) concept descriptions. The notion of constraint graphs [8] captures the structure of sets of constraints.

Definition 2: The labeled graph $g(\Sigma,\Omega)=(\gamma,\delta,\kappa)$ that *captures* Σ and Ω, where κ labels each node with a concept description, is defined as follows:

(i) For each concept description e that occurs on the right- or left-hand side of an inclusion in Σ, or that occurs in Ω, there is exactly one node in γ labeled with e. If necessary, the set of nodes is augmented with new nodes so that:
  (a) For each atomic concept C, there is one node in γ labeled with C.
  (b) For each atomic role P, there is one node in γ labeled with $(\geq 1\ P)$ and one node labeled with $(\geq 1\ P^{-})$.

(ii) If there is a node in γ labeled with a concept description e, then there must be exactly one node in γ labeled with $\overline{e}$.

(iii) For each inclusion $e \sqsubseteq f$ in Σ, there is an arc (M,N) in δ, where M and N are the nodes labeled with e and f, respectively.

(iv) If there are nodes M and N in γ labeled with $(\geq m\ p)$ and $(\geq n\ p)$, where p is either P or P^{-} and $m<n$, then there is an arc (N,M) in δ.

(v) If there is an arc (M,N) in δ, where M and N are the nodes labeled with e and f respectively, then there is an arc (K,L) in δ, where K and L are the nodes labeled with $\overline{f}$ and $\overline{e}$, respectively.

(vi) These are the only nodes and arcs of $g(\Sigma)$. □

Definition 3: The *constraint graph* for Σ and Ω, or the graph that *represents* Σ and Ω, is the labeled graph $G(\Sigma,\Omega)=(\eta,\varepsilon,\lambda)$, where λ labels each node with a set of concept descriptions, defined from $g(\Sigma,\Omega)$ by collapsing each strongly connected component of $g(\Sigma,\Omega)$ into a single node labeled with the descriptions that previously labeled the nodes in the strongly connected component. When Ω is the empty set, we simply write $G(\Sigma)$ and say that $G(\Sigma)$ is the *constraint graph* for Σ. □

If a node K of $G(\Sigma,\Omega)$ is labeled with e, then $\overline{K}$ denotes the node labeled with $\overline{e}$, and $K \rightarrow M$ indicates that there is a path in $G(\Sigma,\Omega)$ from K to M.

Definition 4: Let $G(\Sigma,\Omega)=(\eta,\varepsilon,\lambda)$ be the constraint graph for Σ and Ω. We say that a node K of $G(\Sigma,\Omega)$ is a $\perp$-*node with level* n, for a non-negative integer n, iff one of the following conditions holds:

(i) K is is a $\perp$-*node with level 0* iff
  a. K is labeled with $\perp$, or
  b. there are nodes M and N, not necessarily distinct from K, and a basic concept description h such that M and N are labeled with h and $\neg h$, and $K \rightarrow M$ and $K \rightarrow N$.

(ii) K is a $\bot$-*node with level* $n+1$ iff

a. There is a $\bot$-node M of level n, distinct from K, such that $K \rightarrow M$, and M is the $\bot$-node with the smallest level such that $K \rightarrow M$, or

b. K is labeled with a minCardinality constraint of the form $(\geq 1\ P)$ (or of the form $(\geq 1\ P^{-})$) and there is a $\bot$-node M of level n, distinct from K, such that M is labeled with $(\geq 1\ P^{-})$ (or with $(\geq 1\ P)$), and M is the $\bot$-node with the smallest level labeled with $(\geq 1\ P^{-})$ or $(\geq 1\ P)$. □

Definition 5: Let $G(\Sigma,\Omega)=(\eta,\varepsilon,\lambda)$ be the constraint graph for Σ and Ω. Let K be a node of $G(\Sigma,\Omega)$. We say that K is a $\bot$-*node* iff K is a $\bot$-node with level n, for some non-negative integer n. We also say that K is a $\top$-*node* iff $\overline{K}$ is a $\bot$-node. □

Theorem 1 shows how to test logical implication for $DL\text{-}Lite_{core}^{\mathcal{N}}$ inclusions [8].

Theorem 1. Let Σ be a set of $DL\text{-}Lite_{core}^{\mathcal{N}}$ inclusions and σ be a $DL\text{-}Lite_{core}^{\mathcal{N}}$ inclusion. Assume that σ is of the form $e \sqsubseteq f$ and let $\Omega = \{e, f\}$. Then, $\Sigma \models \sigma$ iff one of the following conditions holds:

(i) The node of $G(\Sigma,\Omega)$ labeled with e is a $\bot$-node; or

(ii) The node of $G(\Sigma,\Omega)$ labeled with f is a $\top$-node; or

(iii) There is a path in $G(\Sigma,\Omega)$ from the node labeled with e to the node labeled with f.

Figure 2 introduces the **IMPLIES** procedure to test logical implication for $DL\text{-}Lite_{core}^{\mathcal{N}}$ inclusions, whose soundness and completeness is a direct consequence of Theorem 1.

IMPLIES(Σ, $e \sqsubseteq f$)
input: a set Σ of unabbreviated inclusions and an unabbreviated inclusion $e \sqsubseteq f$
output: "YES - Σ logically implies $e \sqsubseteq f$"
"NO - Σ does not logically imply $e \sqsubseteq f$"

begin Construct $G(\Sigma, \{e, f\})$, the constraint graph for Σ and $\{e, f\}$;
if the node of $G(\Sigma, \{e, f\})$ labeled with e is a $\bot$-node, or
the node of $G(\Sigma, \{e, f\})$ labeled with f is a $\top$-node, or
there is a path in $G(\Sigma, \{e, f\})$ from the node labeled with e to the node labeled with f,
then return "YES - Σ logically implies $e \sqsubseteq f$";
else return "NO - Σ does not logically imply $e \sqsubseteq f$";
end

Fig. 2. The **IMPLIES** procedure

Example 1: The fragment of the Music Ontology shown in Figure 1 is formalized as the *Agent-Person Ontology*, $\boldsymbol{APO} = (V_{APO}, \Sigma_{APO})$, where

V_{APO} = { foaf:Agent, foaf:Person, foaf:Group, foaf:Organization, mo:MusicArtist, mo:CorporateBody, mo:SoloMusicArtist, mo:MusicGroup, mo:Label, mo:member_of, foaf:name, xsd:string }

and the set of constraints Σ_{APO} shown in Table 2. Figure 3 depicts the constraint graph $G(\Sigma_{APO})$ for Σ_{APO} (using unabbreviated constraints).

Table 2. Constraints of ***APO*** (unabbreviated form)

Constraint	Informal specification
(≥1 foaf:name) ⊑ foaf:Person	The domain of foaf:name is foaf:Person
(≥1 foaf:name$^-$) ⊑ xsd:string	The range of foaf:name is xsd:string
(≥1 mo:member_of) ⊑ foaf:Person	The domain of mo:member_of is foaf:Person
(≥1 mo:member_of$^-$) ⊑ foaf:Group	The range of mo:member_of is foaf:Group
mo:MusicArtist ⊑ foaf:Agent	mo:MusicArtist is a subset of foaf:Agent
foaf:Group ⊑ foaf:Agent	foaf:Group is a subset of foaf:Agent
foaf:Organization ⊑ foaf:Agent	foaf:Organization is a subset of foaf:Agent
mo:SoloMusicArtist ⊑ foaf:Person	mo:SoloMusicArtist is a subset of foaf:Person
mo:SoloMusicArtist ⊑ mo:MusicArtist	mo:SoloMusicArtist is a subset of mo:MusicArtist
mo:MusicGroup ⊑ mo:MusicArtist	mo:MusicGroup is a subset of mo:MusicArtist
mo:MusicGroup ⊑ foaf:Group	mo:MusicGroup is a subset of foaf:Group
mo:CorporateBody ⊑ foaf:Organization	mo:CorporateBody is a subset of foaf:Organization
mo:Label ⊑ mo:CorporateBody	mo:Label is a subset of mo:CorporateBody
foaf:Person ⊑ ¬foaf:Organization	foaf:Person and foaf:Organization are disjoint

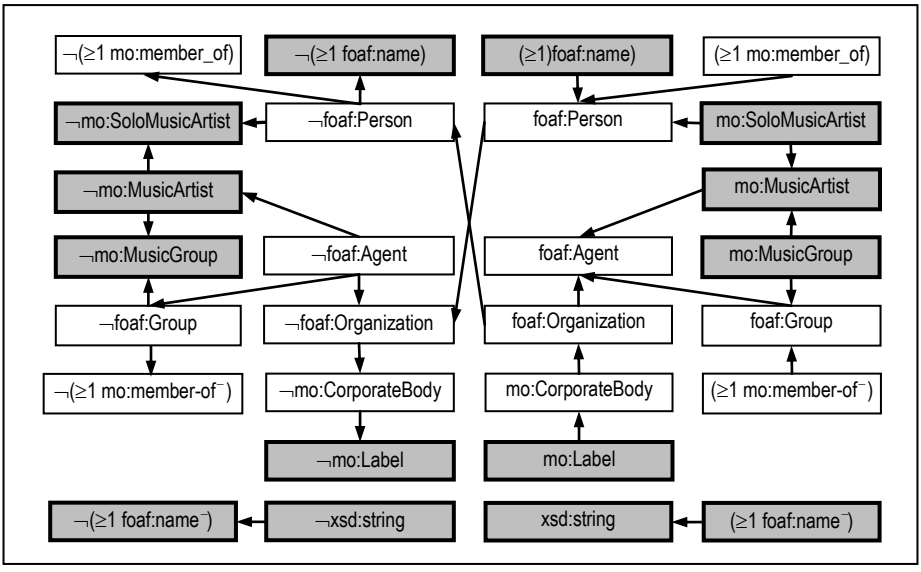

Fig. 3. The constraint graph $G(\Sigma_{APO})$ for Σ_{APO}

3.2 Computing the Operations

In what follows, we say that two ontologies $O_1=(V_1,\Sigma_1)$ and $O_2=(V_2,\Sigma_2)$ are *equivalent* iff $V_1 = V_2$ *and* $\tau[\Sigma_1] = \tau[\Sigma_2]$.

The procedures that compute the operations follow the same pattern. They first compute the transitive closures of the constraint graphs for the constraints of the input ontologies (except for the **Union** operation, which uses the constraint graphs directly). Then, they use the transitive closures to create a graph H that represents a set of constraints. Finally, they use H to generate a set of constraints which is equivalent to the set of constraints of the resulting ontology.

This last step is carried out by the **GenerateConstraints** procedure (in Figure 4), which has three stages: (1) For each $\bot$-node M of H, it generates constraints of the form $e \sqsubseteq \bot$, where e labels M, since all expressions that label a $\bot$-node denote the empty set; (2) For each node M of H which is neither a $\bot$-node nor a $\top$-node, it generates constraints of the form $e \equiv f$, where e and f label M, since all expressions that label the same node are equivalent; (3) For each arc (M,N) of H such that M and N are neither a $\bot$-node nor a $\top$-node, it generates a constraint of the form $e \sqsubseteq f$, where e labels M and f labels N, since an arc denotes an inclusion. Note that we do not create inclusions for all pairs of expressions respectively from M and N, in view of step (2).

Let $O_1=(V_1,\Sigma_1)$ be a lightweight ontology and W be a subset of V_1. Procedure **Projection** (in Figure 5) generates Σ_2 so that $O_2=(W,\Sigma_2)$ is equivalent to the projection of $O_1=(V_1,\Sigma_1)$ over W. **Projection** first constructs the transitive closure $G^*(\Sigma_1)$ of the constraint graph $G(\Sigma_1)$. Then, it uses $G^*(\Sigma_1)$ to create a graph H by discarding all expressions that label nodes of $G^*(\Sigma_1)$ and that do not involve only symbols in W. Finally, it passes H to the **GenerateConstraints** procedure to generate a set of constraints which is equivalent to the set of constraints of the projected ontology.

```
GenerateConstraints(H;Σ)
input:    a constraint graph H
output:   a set of constraints Σ
begin  initialize Σ = ∅;
       for each node M of H such that M is a ⊥-node of H
           for each concept description e such that e labels M do
               add e ⊑ ⊥ to Σ;
       drop all ⊥-nodes and ⊤-nodes from H;
       for each node M of H
           for each pair of concept descriptions e and f such that e and f label M do
               add e ≡ f to Σ;
       for each arc (M,N) of H do
           begin select expressions e and f such that
                       e and f label nodes M and N, respectively, and
                       e ⊑ f is an allowed constraint (in the sense of Section 2), and
                       f̄ ⊑ ē is not already in Σ /* to avoid redundant constraints */
                 add e ⊑ f to Σ;
       return Σ
end
```

Fig. 4. Procedure **GenerateConstraints**

Projection(V_1 , Σ_1 , W ; Σ_2)
input: a vocabulary V_1
a set Σ_1 of normalized constraints over V_1
a subset W of V_1
output: a set of constraints Σ_2
begin **construct** $G(\Sigma_1)$, the constraint graph for Σ_1;
construct $G^*(\Sigma_1)$, the transitive closure of $G(\Sigma_1)$;
construct a constraint graph H by modifying $G^*(\Sigma_1)$ as follows:
drop all labels (i.e., expressions that label nodes)
that use an atomic concept or an atomic role which is not in W;
drop all nodes that have no labels;
GenerateConstraints(H,Σ_2);
return Σ_2
end

Fig. 5. Procedure **Projection**

The above argument can be transformed into a correctness proof of the **Projection** procedure, in the following sense. Let Σ_1/W denote the set of formulas that use only atomic concepts and atomic roles in W and that are logically implied by Σ_1.

Theorem 2: Let $O_1=(V_1,\Sigma_1)$ be an ontology and W be a subset of V_1. Let Σ_2 be the set of constraints which **Projection** outputs for Σ_1 and W. Then, Σ_2 is logically equivalent to Σ_1/W. □

The following example illustrates how the **Projection** procedure operates.

Example 2: Consider the following vocabulary:

V_{MAC} = { mo:MusicArtist, mo:SoloMusicArtist, mo:MusicGroup, mo:Label, foaf:name, xsd:string }

The ontology *Music Artist Contract*, formalized as $\boldsymbol{MAC}=(V_{MAC}, \Sigma_{MAC})$, is defined as the projection of $\boldsymbol{APO} = (V_{APO}, \Sigma_{APO})$, defined in Example 1, over V_{MAC}. The nodes labeled with expressions that use only terms in V_{MAC} are depicted in gray boxes with thicker frames in Figure 3. The **Projection** procedure computes the constraints in Σ_{MAC} shown in Table 3. For example, **Projection** returns the first constraint in Table 3 since $G(\Sigma_{APO})$ has a path from the node labeled with mo:Label to the node labeled with ¬(≥1 foaf:name) (and, hence, $G^*(\Sigma_{APO})$ has an arc between these two nodes). □

Table 3. Constraints of ontology ***MAC*** (unabbreviated form)

Constraint	Informal specification
mo:Label ⊑ ¬(≥1 foaf:name)	$G(\Sigma_{APO})$ has a path from the node labeled with mo:Label to the node labeled with ¬(≥1 foaf:name) (which indicates that a label has no name)
(≥1 foaf:name$^-$) ⊑ xsd:string	The range of foaf:name is xsd:string
mo:SoloMusicArtist ⊑ mo:MusicArtist mo:MusicGroup ⊑ mo:MusicArtist	mo:SoloMusicArtist is a subset of mo:MusicArtist mo:MusicGroup is a subset of mo:MusicArtist
mo:SoloMusicArtist ⊑ ¬mo:Label	mo:SoloMusicArtist and mo:Label are disjoint

Union(V_1 , Σ_1 , V_2 , Σ_2 ; Σ_3)
input: two lightweight ontologies $O_1=(V_1,\Sigma_1)$ and $O_2=(V_2,\Sigma_2)$
output: a set of constraints Σ_3

begin **construct** $G(\Sigma_1)=(\gamma_1,\delta_1,\kappa_1)$ and $G(\Sigma_2)=(\gamma_2,\delta_2,\kappa_2)$; /* constr. graphs for Σ_1 and Σ_2
initialize $H=(\gamma,\delta,\kappa)$, where $\gamma=\gamma_1\cup\gamma_2$, $\delta=\delta_1\cup\delta_2$ and $\kappa=\kappa_1\cup\kappa_2$;
for each pair of nodes M and N of H
such that the intersection of the sets of expressions that label M and N is not empty **do**
modify H by combining M and N into a single node K, which is labeled with all expressions that labeled either M or N;
GenerateConstraints(H, Σ_3);
return Σ_3
end

Fig. 6. Procedure **Union**

Let $O_1=(V_1,\Sigma_1)$ and $O_2=(V_2,\Sigma_2)$ be two lightweight ontologies.

Procedure **Union** (in Figure 6) generates Σ_3 so that $O_3=(V_1\cup V_2,\Sigma_3)$ is equivalent to the union of O_1 and O_2 as follows. It starts by computing $G(\Sigma_1)=(\gamma_1,\delta_1,\kappa_1)$ and $G(\Sigma_2)=(\gamma_2,\delta_2,\kappa_2)$, the constraint graphs of Σ_1 and Σ_2 (we assume that $\gamma_1\cap\gamma_2=\emptyset$). Then, it initializes a graph $H=(\gamma,\delta,\kappa)$, where $\gamma=\gamma_1\cup\gamma_2$, $\delta=\delta_1\cup\delta_2$ and $\kappa=\kappa_1\cup\kappa_2$. Next, it combines two nodes M and N of H into a single node K iff $\kappa(M)\cap\kappa(N)\neq\emptyset$ (i.e., the sets of labels of M and N have an expression in common); furthermore, $\kappa(K)=\kappa(M)\cup\kappa(N)$ (the new node K is labeled with all expressions that originally labeled either M or N). Finally, it passes H to the **GenerateConstraints** procedure to generate Σ_3 so that $O_3=(V_1\cup V_2,\Sigma_3)$ is equivalent to the union of O_1 and O_2.

As already mentioned in section 2.2, selections can be defined as unions. Hence, no specific algorithm for selections will be discussed here.

Procedure **Intersection** (in Figure 7) computes Σ_3 so that $O_3=(V_1\cap V_2,\Sigma_3)$ is equivalent to the intersection of O_1 and O_2. It first computes $G^*(\Sigma_1)=(\gamma^*_1,\delta^*_1,\kappa^*_1)$ and $G^*(\Sigma_2)=(\gamma^*_2,\delta^*_2,\kappa^*_2)$, the transitive closures of the constraint graphs of Σ_1 and Σ_2. Then, it constructs a graph $H=(\gamma,\delta,\kappa)$ such that there is an arc $(M_1,N_1)\in\gamma$ iff there are arcs (M_1,N_1) of $G^*(\Sigma_1)$ and (M_2,N_2) of $G^*(\Sigma_2)$ and expressions e and f such that M_1 and M_2 are both labeled with e and N_1 and N_2 are both labeled with f; furthermore, $\kappa(M_1)=\kappa^*_1(M_1)\cap\kappa^*_2(M_2)$ and $\kappa(N_1)=\kappa^*_1(N_1)\cap\kappa^*_2(N_2)$. Finally, it passes H to the **GenerateConstraints** procedure to generate a set of constraints Σ_3 so that $O_3=(V_1\cap V_2,\Sigma_3)$ is equivalent to the intersection of O_1 and O_2.

Procedure **Difference** (in Figure 8) computes Σ_3 so that $O_3=(V_1,\Sigma_3)$ is equivalent to the difference of O_1 and O_2. It first computes $G^*(\Sigma_1)=(\gamma^*_1,\delta^*_1,\kappa^*_1)$ and $G^*(\Sigma_2)=(\gamma^*_2,\delta^*_2,\kappa^*_2)$. Then, it constructs a graph $H=(\gamma,\delta,\kappa)$ such that there is an arc $(M_1,N_1)\in\gamma$ iff, for any pair of expressions e and f such that M_1 is labeled with e and N_1 is labeled with f, there is no arc (M_2,N_2) of $G^*(\Sigma_2)$ such that M_2 is also labeled with e and N_2 is also labeled with f; furthermore, $\kappa(M_1)=\kappa^*_1(M_1)$ and $\kappa(N_1)=\kappa^*_1(N_1)$. Finally, it passes H to the **GenerateConstraints** procedure to generate Σ_3 so that $O_3=(V_1,\Sigma_3)$ is equivalent to the difference of O_1 and O_2.

Intersection(V_1 , Σ_1 , V_2 , Σ_2 ; Σ_3)
input: two lightweight ontologies $O_1=(V_1,\Sigma_1)$ and $O_2=(V_2,\Sigma_2)$
output: a set of constraints Σ_3
begin **construct** $G(\Sigma_1)=(\gamma_1,\delta_1,\kappa_1)$ and $G(\Sigma_2)=(\gamma_2,\delta_2,\kappa_2)$; /* constr. graphs for Σ_1 and Σ_2
construct $G^*(\Sigma_1)=(\gamma^*_1,\delta^*_1,\kappa^*_1)$ and $G^*(\Sigma_2)=(\gamma^*_2,\delta^*_2,\kappa^*_2)$; /* trans. closures
initialize $H=(\gamma,\delta,\kappa)$, where $\gamma=\emptyset$, $\delta=\emptyset$ and $\kappa=\emptyset$;
for each pair of arcs (M_1,N_1) of $G^*(\Sigma_1)$ and (M_2,N_2) of $G^*(\Sigma_2)$
such that there are expression e and f such that
M_1 and M_2 are both labeled with e and
N_1 and N_2 are both labeled with f **do**
add an arc (M_1,N_1) to H with
$\kappa(M_1)=\kappa^*_1(M_1)\cap\kappa^*_2(M_2)$ and
$\kappa(N_1)=\kappa^*_1(N_1)\cap\kappa^*_2(N_2)$;
GenerateConstraints(H, Σ_3);
return Σ_3
end

Fig. 7. Procedure **Intersection**

Difference(V_1 , Σ_1 , V_2 , Σ_2 ; Σ_3)
input: two lightweight ontologies $O_1=(V_1,\Sigma_1)$ and $O_2=(V_2,\Sigma_2)$
output: a set of constraints Σ_3
begin **construct** $G(\Sigma_1)=(\gamma_1,\delta_1,\kappa_1)$ and $G(\Sigma_2)=(\gamma_2,\delta_2,\kappa_2)$; /* constr. graphs for Σ_1 and Σ_2
construct $G^*(\Sigma_1)=(\gamma^*_1,\delta^*_1,\kappa^*_1)$ and $G^*(\Sigma_2)=(\gamma^*_2,\delta^*_2,\kappa^*_2)$; /* trans. closures
initialize $H=(\gamma,\delta,\kappa)$, where $\gamma=\emptyset$, $\delta=\emptyset$ and $\kappa=\emptyset$;
for each arc (M_1,N_1) of $G^*(\Sigma_1)$
such that for any pair of expressions e and f such that
M_1 is labeled with e and N_1 is labeled with f,
there is no arc (M_2,N_2) of $G^*(\Sigma_2)$ such that
M_2 is also labeled with e and N_2 is also labeled with f **do**
add an arc (M_1,N_1) to H with
$\kappa(M_1)=\kappa^*_1(M_1)$ and $\kappa(N_1)=\kappa^*_1(N_1)$;
GenerateConstraints(H, Σ_3);
return Σ_3
end

Fig. 8. Procedure **Difference**

3.3 Optimizing the Representation of the Resulting Lightweight Ontologies

The procedures that implement each of the operations, outlined in Section 3.2, return a set of constraints that may contain redundancies. This follows because all such procedures in fact work with the transitive closure of the constraint graph (c.f. the

procedure in Figure 4). Therefore, an interesting question immediately arises: how to minimize the set of constraints of an ontology.

We argue that this question is equivalent to finding the *minimum equivalent graph (MEG)* of a graph G, defined as the graph G' with the minimum set of edges such that the transitive closure of G and G' are equal. This problem has a polynomial solution when G is acyclic and is NP-hard for strongly connected graphs [1][13][16].

Let $O_1=(V_1,\Sigma_1)$ be a lightweight ontology and $G(\Sigma_1)=(\eta,\varepsilon,\lambda)$ be the constraint graph for Σ_1. In what follows, we outline how to construct a new set of constraints, Σ_2, such that Σ_2 is a minimal set and Σ_1 and Σ_2 are tautologically equivalent.

Recall that $G(\Sigma_1)$ is acyclic and that each node of $G(\Sigma_1)$ is labeled with a set of expressions that are equivalent.

Since $G(\Sigma_1)$ is acyclic, we may construct a MEG $G'=(\eta,\varepsilon,\lambda)$ of $G(\Sigma_1)$, in polynomial time [1][13]. The nodes of G' are those of G, with the same labels as in G. Note that the procedure adopted to construct a MEG of a graph has to be modified to also drop an arc (M,N), if the arc $(\overline{N},\overline{M})$ connecting the dual nodes of M and N is dropped. This modification is necessary to preserve the characteristics of constraint graphs (see Definitions 2 and 3). Then, for each arc (M,N) in ε', select an expression e that labels M and an expression f that labels N, and add the constraint $e \sqsubseteq f$ to Σ_2.

The apparent problem lies in the expressions that label each node of $G(\Sigma_1)$. Indeed, by Definition 3, $G(\Sigma_1)$ is defined from $g(\Sigma_1)$ by collapsing each strongly connected component of $g(\Sigma_1)$ into a single node labeled with the descriptions that previously labeled the nodes in the strongly connected component. Thus, we would have to construct a MEG, G'', of each strongly connected component of $g(\Sigma_1)$ and generate a set of constraints $e \sqsubseteq f$ from G'' as before. However, constructing a MEG of a strongly connected graph is NP-hard (albeit there are good approximation algorithms [16]).

However, recall that, for any two expressions e and f that label the same node of $G(\Sigma_1)$, we have that Σ_1 logically implies $e \equiv f$. From the point of view of an economical ontology design, for each strongly connected component of $g(\Sigma_1)$, we suggest to construct a minimal set of equivalences which are tautologically equivalent to the inclusions that correspond to the arcs of $g(\Sigma_1)$.

But this new problem is equivalent to construct a spanning tree T for each strongly connected component of $g(\Sigma_1)$, which can be done in polynomial time. Next, for each edge $\{M,N\}$ of T, add $e \equiv f$ to Σ_2, where e labels M and f labels N in $g(\Sigma_1)$. If T has k nodes, we will then generate $k-1$ equivalences.

The final set of constraints, Σ_2, will contain a minimal set of inclusions, which correspond to the arcs of $G(\Sigma_1)$, and a minimal set of equivalences, generated from the expressions that label the nodes of each strongly connected component of $g(\Sigma_1)$. Finally, by construction, Σ_1 and Σ_2 will be tautologically equivalent. Furthermore, we may argue that the equivalences better capture the semantics of the expressions that label each strongly connected component of $g(\Sigma_1)$, as compared with trying to generate inclusions from a MEG of the strongly connected component.

4 The OntologyManagement Tool

In this section, we briefly outline the main functionalities of the ***OntologyManagement*** tool, which we developed to experiment with the operations introduced in previous sections.

Figure 9 depicts a typical scenario illustrating the basic functionalities of this tool. First, in step 1, the user chooses the ontologies that will serve as operands of the selected operation. The tool allows loading ontologies written in OWL. Next, in step 2, the tool constructs the constraint graphs corresponding to the selected ontologies, as discussed in Section 3.2. Then, in step 3, the user selects an operation. Finally, in step 4, the tool executes the selected operation using the constraint graphs.

Figure 10 shows the graphical user interface of the ***OntologyManagement*** tool (with two ontologies loaded). This interface presents each constraint graph within a two-column table, where each line represents an inclusion of the form $e \sqsubseteq f$, where e and f are both basic concept descriptions, shown in the first and second column, respectively. The two leftmost tables present the operand ontologies that are used by the selected operation to compute the resulting ontology, shown in the third table. Finally, the resulting ontology may be saved as a new OWL ontology, allowing the user to utilize it in another ontology operation.

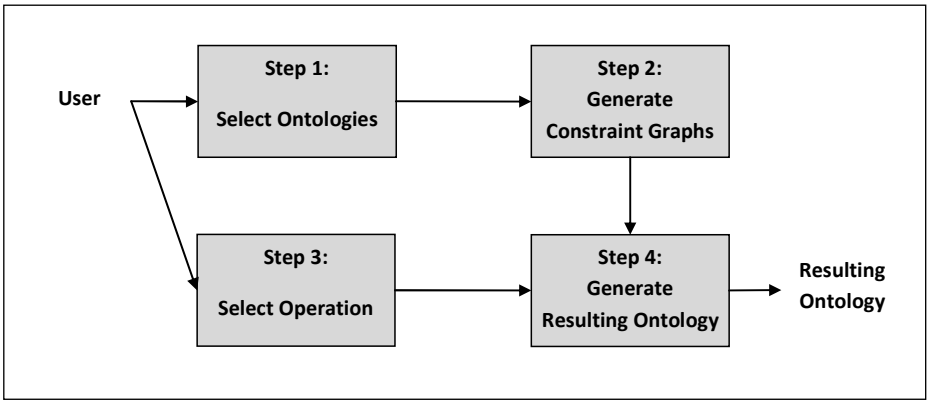

Fig. 9. The architecture of the ***OntologyManagement*** tool

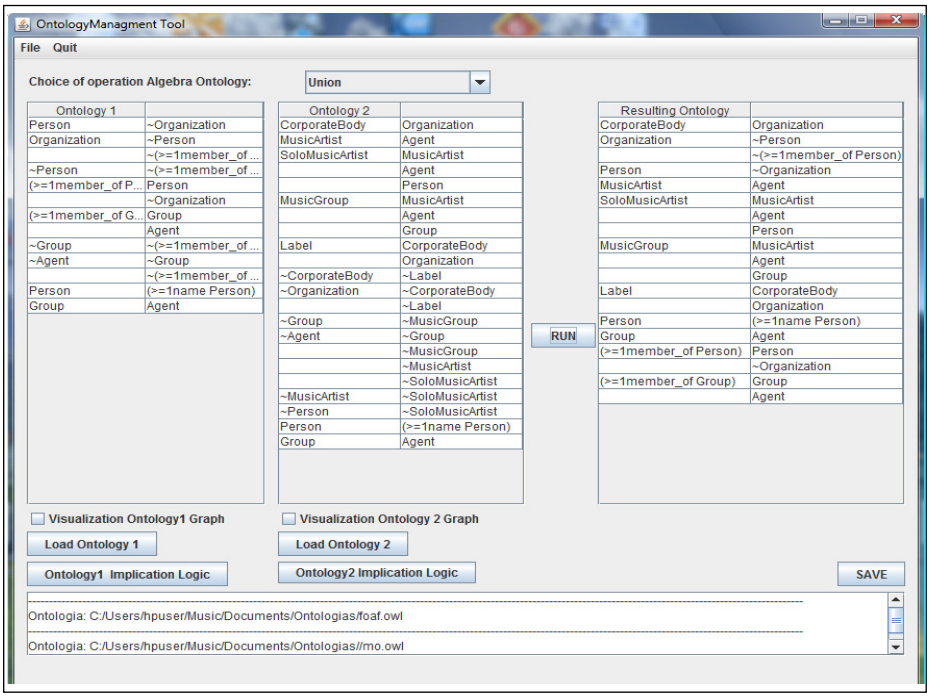

Fig. 10. The user interface of the ***OntologyManagement*** tool

5 Related Work

The results reported in the paper cover a topic – improving Linked Data design by constraint reuse – that is still neglected in the literature. The question of Linked Data semantics is not new, though. For example, recent investigation [12][14][17] in fact questions the correct use of `owl:sameAs` to inter-link datasets.

The results contribute to the discussion on the mapping process from relational databases (RDBs) to RDF [11]. Indeed, RDB-to-RDF tools (see [20] for a comprehensive survey) typically limit themselves to support vocabulary reuse, if at all. We argued that RDB-to-RDF tools should go further and include an analysis of the constraints of the domain ontology that apply to the data being published since such constraints capture the semantics of the reused terms. Furthermore, they may include the optimization of the obtained sets of constraints.

Jain et al. [15] argues that the Linked Open Data (LoD) Cloud, in its current form, is only of limited value for furthering the Semantic Web vision. They discuss that the Linked Open Data Cloud can be transformed from "merely more data" to "semantically linked data" by overcoming problems such as lack of conceptual descriptions for the datasets, schema heterogeneity and absence of schema level links. Along this line, we advocated that the design of Linked Data sources must include constraints derived from those of the underlying ontologies.

We note that the problem we cover in this paper cannot be reduced to a question of ontology alignment in the context of Linked Data, addressed for example in [18][21]. Indeed, we stress that the problem we focus on refers to bootstrapping a new ontology (including its constraints) through the implementation of ontology algebra operations (selection, projection, union, intersection and difference) over one or more existing ontologies.

Finally, previous work by the authors [9] introduced the notion of *open fragment*, which is captured by the projection operation. However, we stress, that none of the previous work by the authors considered the selection, union and difference operations, the question of optimizing the representation of the resulting constraints or described a tool to implement the ontology operations. These are original contributions of this paper. Furthermore, by considering the union operation, we relax the assumption adopted in [9] that the application ontology is defined over a single domain ontology. We also included in the same conceptual framework the intersection operation, used in [8] to model the constraints of a mediated schema.

6 Conclusions

In this paper, we introduced a set of concepts and procedures that promote constraint reuse in Linked Data design. We defined a set of operations that extend the idea of namespaces to take into account constraints and treat ontologies as theories. We showed how to compute the operations when the ontologies are built upon DL-Lite core with arbitrary number restrictions. For such ontologies, we also showed how to minimize the set of constraints that result from an operation.

As for current work, we are also designing a tool to extract constraints from a Linked Data source $\boldsymbol{S}$ by combining the information in the VOID document of the source, if any, with a probing technique. The tool explores the idea of the operations introduced in this paper both to compute the constraints that apply to $\boldsymbol{S}$ and to document them in an extension of the VOID vocabulary.

As for future work, we intend to integrate our management tool with the Protégé ontology editor. The objective of this integration is to take advantage of all functionalities already available in Protégé, such as ontology modeling and visualization, inference and reasoning tasks.

Acknowledgements. This work was partly supported by CNPq, under grants 301497/2006-0, 557128/2009-9, 483552/2009-7, 308247/2009-4, 475717/2011-2 and 552578/2011-8, by FAPERJ under grants E-26/170028/2008 and E-26/103.070/2011, by CAPES under grant NF 21/2009, and by FUNCAP under grant GPF 2151/22.

References

1. Aho, A.V., Garey, M.R., Ullman, J.D.: The Transitive Reduction of a Directed Graph. SIAM J. Comp. 1(2), 131–137 (1972)
2. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-Lite family and relations. J. of Artificial Intelligence Research 36(1), 1–69 (2009)

3. Baader, F., Nutt, W.: Basic Description Logics. In: Baader, F., Calvanese, D., McGuiness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.) The Description Logic Handbook: Theory, Implementation and Applications, pp. 43–95. Cambridge U. Press, Cambridge (2003)
4. Berners-Lee, T.: Linked Data - Design Issues. W3C (July 2006)
5. Bizer, C., Cyganiak, R., Heath, T.: How to publish Linked Data on the Web (July 2007), http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/
6. Bizer, C., Heath, T., Berners-Lee, T.: Linked Data - The Story So Far. Int. Journal on Semantic Web and Information Systems 5(3), 1–22 (2009)
7. Brickley, D., Miller, L.: FOAF Vocabulary Specification 0.98. Namespace Document 9 Marco Polo Edition (August 2010)
8. Casanova, M.A., Lauschner, T., Leme, L.A.P.P., Breitman, K.K., Furtado, A.L., Vidal, V.M.P.: Revising the Constraints of Lightweight Mediated Schemas. Data & Knowledge Engineering 69(12), 1274–1301 (2010)
9. Casanova, M.A., Beitman, K.K., Furtado, A.L., Vidal, V.M.P., Macedo, J.A.F., Gomes, R.V.A., Salas, P.E.R.: The Role of Constraints in Linked Data. In: Meersman, R., Dillon, T., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.-C., Damiani, E., Schmidt, D.C., White, J., Hauswirth, M., Hitzler, P., Mohania, M. (eds.) OTM 2011, Part II. LNCS, vol. 7045, pp. 781–799. Springer, Heidelberg (2011)
10. Casanova, M.A., Breitman, K.K., Furtado, A.L., Vidal, V.M.P., Macêdo, J.A.F.: An Efficient Proof Procedure for a Family of Lightweight Database Schemas. In: Hinchey, M.G. (ed.) Conquering Complexity, pp. 431–461. Springer (2012)
11. Das, S., Sundara, S., Cyganiak, R.: R2rml: RDB to RDF mapping language. W3C RDB2RDF working group, http://www.w3.org/TR/r2rml/
12. Halpin, H., Hayes, P.J.: When owl:sameAs isn't the same: An analysis of identity links on the semantic web. In: Proc. Int'l. Workshop on Linked Data on the Web (2010)
13. Hsu, H.T.: An Algorithm for Finding a Minimal Equivalent Graph of a Digraph. Journal of the Association for Computing Machinery 22(1), 11–16 (1975)
14. Jaffri, A., Glaser, H., Millard, I.: URI disambiguation in the context of linked data. In: Proc. 1st Int'l. Workshop on Linked Data on the Web (2008)
15. Jain, P., Hitzler, P., Yeh, P.Z., Verma, K., Sheth, A.P.: Linked Data is Merely More Data. In: Proc. AAAI Spring Symp. Linked Data Meets Artificial Intelligence, pp. 82–86 (2010)
16. Khuller, S., Raghavachari, B., Young, N.: Approximating the Minimum Equivalent Digraph. SIAM Journal on Computing 24(4), 859–872 (1995)
17. McCusker, J., McGuinness, D.L.: owl:sameAs considered harmful to provenance. In: Proc. ISCB Conference on Semantics in Healthcare and Life Sciences (2010)
18. Jain, P., Hitzler, P., Sheth, A.P., Verma, K., Yeh, P.Z.: Ontology Alignment for Linked Open Data. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 402–417. Springer, Heidelberg (2010)
19. Raimond, Y., Giasson, F.: Music Ontology Specification. Specification Document (RDF/XML, Turtle) (November 28, 2010), http://purl.org/ontology/mo/
20. Sahoo, S.S., Halb, W., Hellmann, S., Idehen, K., Thibodeau Jr, T., Auer, S., Sequeda, J., Ezzat, A.: A survey of current approaches for mapping of relational databases to rdf. W3C RDB2RDF Incubator Group Report (2009)
21. Wang, Z., Zhang, X., Hou, L., Li, J.: RiMOM2: A Flexible Ontology Matching Framework. In: Proc. ACM WebSci 2011, Koblenz, Germany, pp. 1–2 (2011)