

The Role of Constraints in Linked Data

Marco Antonio Casanova¹, Karin Koogan Beitman¹, Antonio Luz Furtado¹,
Vania M.P. Vidal², José A.F. Macedo², Raphael Valle A. Gomes¹,
and Percy E. Rivera Salas¹

¹Department of Informatics – PUC-Rio – Rio de Janeiro, RJ – Brazil

{casanova, karin, furtado, rgomes, psalas}@inf.puc-rio.br

²Department of Computing, Federal University of Ceará – Fortaleza, CE – Brazil

{vvidal, jose.macedo}@lia.ufc.br

Abstract. The paper argues that a Linked Data source should publish an application ontology that includes a set of constraints that capture the semantics of the classes and properties used to model the data. Furthermore, if the Linked Data source publishes a mapping between its vocabulary and the vocabulary of a domain ontology, then it has to specify the application ontology constraints so that they are consistent with those of the domain ontology. The main contributions of the paper are methods for constructing the constraints of the application ontology of a Linked Data source, defined as fragments of domain ontologies. The methods assume that the ontologies are written in an expressive family of attributive languages and depend on a procedure to test logical implication, which explores the structure of sets of constraints.

Keywords: constraints, application ontology, Linked Data.

1 Introduction

The term *Linked Data* refers to a set of best practices for publishing and connecting structured data on the Web [4]. A Linked Data source may publish an *application ontology* that models the exported data and a mapping between the application ontology vocabulary and a *domain ontology* vocabulary (or several such vocabularies). The mapping may be expressed as a set of RDF triples that link classes and properties in one vocabulary to those in another, or it may be defined using a schema mapping language.

In this paper, we argue that a Linked Data source should include, in the definition of the application ontology, a set of constraints that capture the semantics of the classes and properties used to model the data. Furthermore, if the Linked Data source publishes a mapping between its vocabulary and the vocabulary of the domain ontology, then it has to specify the application ontology constraints so that they are consistent with those of the domain ontology.

More precisely, an *ontology* is a pair $O=(V_O, C_O)$, where V_O is a vocabulary and C_O is a set of constraints over V_O . A *domain ontology* $D=(V_D, C_D)$ models the application domain. In fact, D may be a combination of ontologies covering distinct domains. An *application ontology* $A=(V_A, C_A)$ models the data exported by a Linked Data source.

The problem we address can be formulated as follows: “Given that V_A is a subset of V_D , how to derive C_A from C_D ”. We offer two alternative answers to this question, depending on what we require from the data exported by the data source.

Suppose first that we require that the data exported by the Linked Data source must satisfy C_A , and that C_A must logically imply all constraints that can be derived from C_D and that use only symbols in V_A . In this case, we say that the application ontology is an *open fragment* of the domain ontology. Section 4 formulates these requirements in detail and describes a method to derive C_A .

Suppose now that we require that the data exported by the Linked Data source must satisfy C_D , when all classes and properties in V_D , but not in V_A , are taken as the empty set (when the source data is published). In this case, we say that the application ontology is a *closed fragment* of the domain ontology. Section 5 addresses this case.

Applications may benefit from these concepts as follows. Consider a Linked Data source S whose data is published according to an application ontology \mathcal{A} . Suppose first that \mathcal{A} is designed as an open fragment of \mathcal{D} . Then, in general, any application that processes data modeled according to \mathcal{D} and *that uses only the classes and properties in the vocabulary of \mathcal{A}* will also be able to process data published by S (since the application expects data consistent with the constraints derived from C_D that apply to the classes and properties in the vocabulary of \mathcal{A}). Now, suppose that \mathcal{A} is designed as a closed fragment of \mathcal{D} . Then, any application that processes data modeled according to \mathcal{D} will also be able to process data published by S (since the application expects data consistent with the constraints in C_D).

In particular, consider a query optimizer that wishes to submit a query Q to the data source S . Again, suppose first that \mathcal{A} is designed as an open fragment of \mathcal{D} . Then, if Q uses only the classes and properties in the vocabulary of \mathcal{A} , the optimizer needs to consider only the constraints derived from C_D that apply to the classes and properties in the vocabulary of \mathcal{A} . Indeed, any data from S will satisfy such constraints since \mathcal{A} was designed as an open fragment of \mathcal{D} . Any other constraint derived from C_D will be irrelevant to the process. Suppose now that \mathcal{A} is designed as a closed fragment of \mathcal{D} . Then, the optimizer needs to consider the constraints in C_D , but it may also assume that any class or property not in the vocabulary of \mathcal{A} is empty (by definition of closed fragment). This opens new opportunities for relatively straightforward optimizations.

The main contributions of the paper are methods for constructing application ontology constraints when the application ontology is an open or a closed fragment of the domain ontology. The methods assume that the ontologies are written in an expressive family of attributive languages and depend on a procedure to test logical implication, which explores the structure of sets of constraints, captured as *constraint graphs* [7]. The methods are also useful in other application domains, such as data integration and data mashups, where a set of constraints have to be constructed from other sets of constraints.

The paper is organized as follows. Section 2 further discusses the motivation for the paper. Section 3 presents the formal framework adopted in the paper. Section 4 focuses on how to construct open fragments of the domain ontology. Section 5 analyses the case of closed fragments. Section 6 summarizes related work. Finally, Section 7 contains the conclusions.

2 An Informal Example

The ‘Linked Data Principles’ [2] 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 [5]:

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.

We are particularly interested in the second characteristic. The definition of vocabulary terms ultimately includes 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 ontology from which he is drawing the terms to construct his vocabulary. We further motivate this argument with the help of examples, adopting the *Music Ontology (MO)* [17] as the domain ontology.

The Music Ontology is used by several Linked Data sources, including MusicBrainz and BBC Music. The Music Ontology RDF schema uses terms from the *FRBR* [14], *FOAF* [6] and the XML Schema vocabularies. We adopt the prefixes “mo:”, “frbr:”, “foaf:” and “xsd:” to respectively refer to the *MO*, *FRBR*, *FOAF* and XML Schema vocabularies.

Fig. 1 shows the class hierarchies of *MO* rooted at classes `event:Event` and `frbr:Expression`. Fig. 2 shows the class hierarchies of *MO* rooted at classes `foaf:Agent` and `foaf:Person`.

Suppose that the designer wants to publish a dataset, using *MO* as the domain ontology. He then proceeds to define an application ontology, which we call *Signal ontology (SGL)*. As the first step, he selects classes and properties from the *MO* vocabulary to create the *SGL* vocabulary. Assume that he selects classes `mo:Signal`, `mo:DigitalSignal` and `mo:analogSignal`, the datatype property `mo:isrc` and the object property `mo:sampled_version` to form the *SGL* vocabulary (all shown in Fig. 1).

Usually, strategies to publish Linked Data treat domain ontology vocabularies only up to this stage. We argue that the strategies should go further and include an analysis of the constraints of the domain ontology that apply to the data being published.

By observing the *MO* constraints (informally depicted in Fig. 1), the designer may directly derive the following constraints for the application ontology *SGL*:

- `mo:DigitalSignal` and `mo:analogSignal` are subclasses of `mo:Signal`
- `mo:DigitalSignal` and `mo:analogSignal` are disjoint classes
- the domain and range of `mo:sampled_version` are `mo:analogSignal` and `mo:DigitalSignal`, respectively
- the domain and range of `mo:isrc` are `mo:Signal` and `xsd:String`, respectively

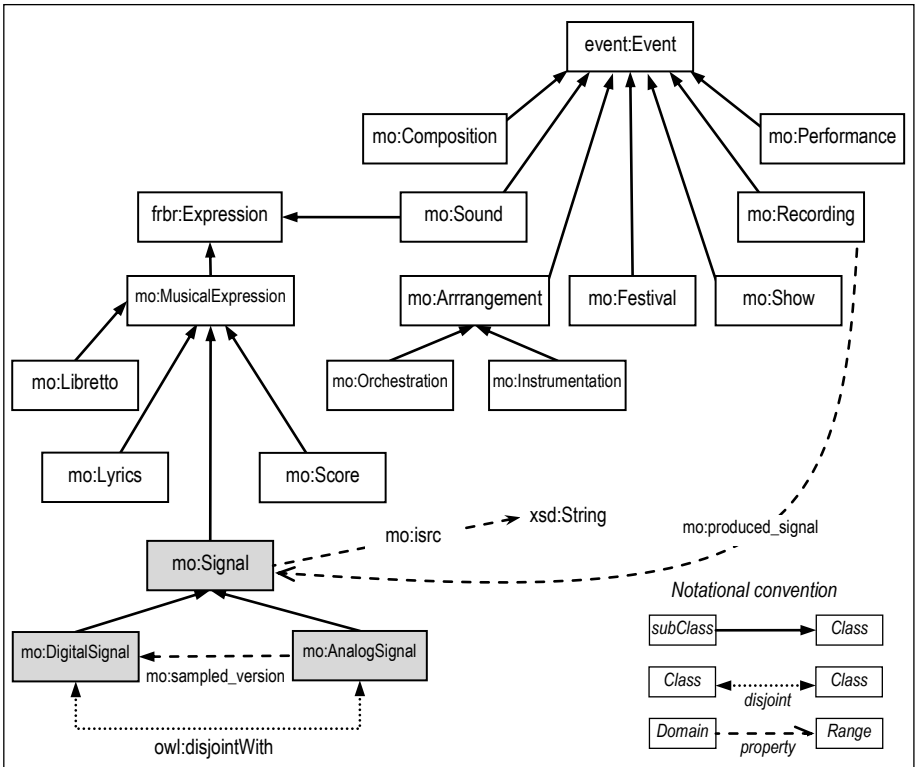


Fig. 1. The class hierarchies of *MO* rooted at classes event:Event and frbr:Expression

The constraints of the application ontology *SGL* were straightforward to obtain because they were exactly the constraints of *MO* that involve the classes and properties in the *SGL* vocabulary. Hence, if the designer publishes his data source so that the set of triples satisfies the *SGL* constraints, then any Web application that processes data modeled according to *MO* and that uses only the classes and properties in the *SGL* vocabulary will also be able to process the triples published by the data source. This follows because the application expects data consistent with the *MO* constraints that apply to the classes and properties it uses.

Suppose that the designer wants to publish a second dataset, again using *MO* as the domain ontology. He proceeds to define an application ontology, which we call *Artist Contract (AC)*. Assume that he selects classes mo:SoloMusicArtist, mo:MusicGroup and mo:label, and the object property mo:member_of to create the *AC* vocabulary (all shown in Fig. 2).

The derivation of the *AC* constraints is not as straightforward as before since *MO* has no constraints involving just the terms in the *AC* vocabulary. However, observe from Fig. 2 that foaf:Person and foaf:Organization are disjoint classes. Therefore, the designer may infer that their subclasses, mo:SoloMusicArtist and mo:Label, respectively, are also disjoint. This constraint must therefore be in the set of *AC* constraints. Note that this constraint is inferred from, but not a member of the set of *MO* constraints.

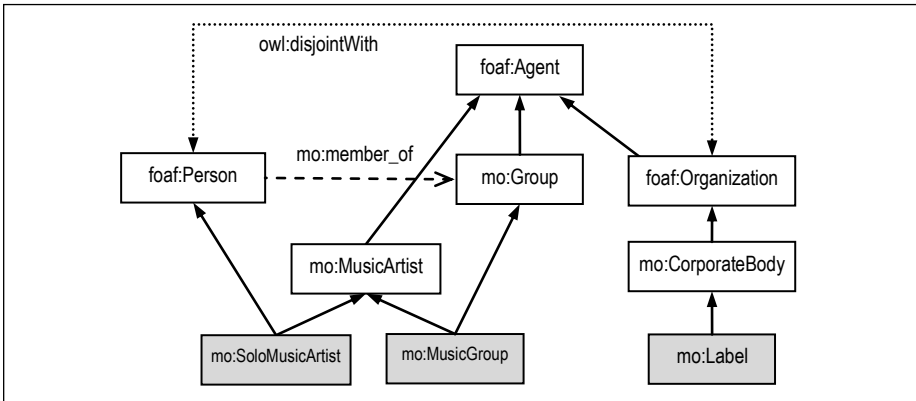


Fig. 2. The class hierarchies of *MO* rooted at classes *foaf:Agent* and *foaf:Person*

The question of including *mo:member_of* in the *AC* vocabulary, without including the original classes *foaf:Person* and *mo:Group* used to define its domain and range, raises a different question. One alternative is to ignore domain and range constraints when designing the *AC* constraints. A second alternative is to inform the designer that he should also include *foaf:Person* and *mo:Group* in the *AC* vocabulary. Sections 4 and 5 discuss these two alternatives in depth.

These informal examples illustrate that the domain ontology constraints play an essential role when designing the application ontology since they carry the semantics of the terms in the domain ontology vocabulary. They also raise the question that the design process cannot be reduced to merely copying the constraints from the domain ontology to the application ontology. We must take into account constraints derived from those of the domain ontology. This point is addressed in the rest of the paper.

3 A Formal Framework

3.1 A Brief Review of Attributive Languages

The reader may wish to skip this section on a first reading and go directly to Section 3.2 that introduces the notion of extralite ontologies and that has an informal description of the constraint semantics.

We adopt a family of *attributive languages* [1] defined as follows. A *language* \mathcal{L} in the family is characterized by an *alphabet* \mathbf{A} , consisting of a set of *atomic concepts*, a set of *atomic roles*, the *universal concept* \top and the *bottom concept* \perp . The set of *role descriptions* and the set of *concept descriptions* of \mathcal{L} (or in \mathbf{A}) are defined as follows:

- An atomic concept, and the universal and bottom concepts are concept descriptions, and an atomic role is a role description

- If e and f are concept descriptions and p is a role description, then $\neg e$ (*negation*), $e \sqcup f$ (*union*), and $(\geq n p)$ (*at-least restriction*) are concept descriptions, and p^- (*inverse*) is a role description.

An *interpretation* s for \mathcal{A} consists of a nonempty set Δ^s , the *domain* of s , whose elements are called *individuals*, and an *interpretation function*, also denoted s , where:

- $s(\perp) = \emptyset$ and $s(\top) = \Delta^s$
- $s(A) \subseteq \Delta^s$, for each atomic concept A of \mathcal{A}
- $s(P) \subseteq \Delta^s \times \Delta^s$, for each atomic role P of \mathcal{A}

The function s is extended to role and concept descriptions of \mathcal{L} as follows:

- $s(\neg e) = \Delta^s - s(e)$ (the complement of $s(e)$ w.r.t. Δ^s)
- $s(e \sqcup f) = s(e) \cup s(f)$ (the union of $s(e)$ and $s(f)$)
- $s(\geq n p) = \{I \in \Delta^s \mid |\{J \in \Delta^s \mid (I, J) \in s(p)\}| \geq n\}$
(the set of individuals that $s(p)$ relates to at least n distinct individuals)
- $s(p^-) = s(p)^-$ (the inverse of $s(p)$)

A *formula* of \mathcal{L} (or in \mathcal{A}) is an expression of the form $u \sqsubseteq v$, called an *inclusion*, or of the form $u \equiv v$, called an *equivalence*, where u and v are both concept descriptions or they are both role descriptions of \mathcal{L} . A *definition* is an equivalence of the form $D \equiv e$, where D is an atomic concept and e is a concept description, or D is an atomic role and e is a role description.

Let s be an interpretation for \mathcal{A} , σ be a formula and Σ and Γ be sets of formulas of \mathcal{L} . We say that

- s satisfies $u \sqsubseteq v$ iff $s(u) \subseteq s(v)$, and s satisfies $u \equiv v$ iff $s(u) = s(v)$
- s is a *model* of Σ , denoted $s \models \Sigma$, iff s satisfies all formulas in Σ
- Σ *logically implies* σ , denoted $\Sigma \models \sigma$, iff any model of Σ satisfies σ
- Σ *logically implies* Γ , denoted $\Sigma \models \Gamma$, iff any model of Σ is also a model of Γ

If \mathcal{B} is a subset of \mathcal{A} , then Σ / \mathcal{B} denotes the set of formulas σ that use only symbols in \mathcal{B} and that are logically implied by Σ .

In the next sections we will also use the following abbreviations: “ $e \sqcap f$ ” (*intersection*) for “ $\neg(\neg e \sqcup \neg f)$ ”, “ $\exists p$ ” (*existential quantification*) for “ $(\geq 1 p)$ ”, “ $(\leq n p)$ ” (*at-most restriction*) for “ $\neg(\geq n+1 p)$ ” and “ $u \mid v$ ” (*disjunction*) for “ $u \sqsubseteq \neg v$ ”.

3.2 Extralite Ontologies

We will work with *extralite ontologies* [7] that partially correspond to OWL Lite.

Definition 1

- (a) A *strict extralite ontology* is a pair $\mathcal{O} = (V_{\mathcal{O}}, C_{\mathcal{O}})$ such that

- (i) V_O is a finite alphabet, called the *vocabulary* of O , whose atomic concepts and atomic roles are called the *classes* and *properties* of O , respectively.
 - (ii) C_O is a set of formulas in V_O , called the *constraints* of O , which must be of one the forms shown in Fig. 3.
 - (iii) For each property P in V_O , there is at least one domain and one range constraint for P in C_O .
- (b) A *non-strict extralite ontology* is a pair $O=(V_O, C_O)$ that satisfies only conditions (i) and (ii) above.
- (c) An *extralite ontology* is either a strict or a non-strict extralite ontology. \square

Fig. 3 introduces the constraint types allowed in extralite ontologies and informally defines their semantics, recalling that a class denotes a set of individuals and a property denotes a set of pairs of individuals. Fig. 3 also shows the *unabbreviated form* of a constraint. Note that a constraint and its unabbreviated form are equivalent. For example, the unabbreviated form of “ $C \mid D$ ” is “ $C \sqsubseteq \neg D$ ”.

Finally, a *constraint expression* is an expression that may occur on the right- or left-hand sides of an unabbreviated constraint.

Constraint Type	Formalization	Unabbreviated form	Informal semantics
Domain Constraint	$\exists P \sqsubseteq D$	$(\geq 1 P) \sqsubseteq D$	property P has class D as domain, that is, if (a, b) is a pair in P , then a is an individual in D
Range Constraint	$\exists P^- \sqsubseteq R$	$(\geq 1 P^-) \sqsubseteq R$	property P has class R as range, that is, if (a, b) is a pair in P , then b is an individual in R
minCardinality Constraint	$C \sqsubseteq (\geq k P)$ 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	$E \sqsubseteq F$		each individual in E is also in F , that is, class E denotes a subset of class F
Disjointness Constraint	$E \mid F$	$C \sqsubseteq \neg D$	no individual is in both E and F , that is, classes E and F are disjoint

Fig. 3. Extralite constraints

3.3 Constraint Graphs

The notion of concept graphs captures the structure of sets of constraints and is essential to the constraint construction methods of Sections 4 and 5. Again, the reader may wish to skip this section on a first reading and go directly to Section 3.4 that contains self-contained examples of constraint graphs.

We say that the *complement* of a non-negated expression e is $\neg e$, and vice-versa; the *complement* of \perp is \top , and vice-versa. If c is an expression, then \bar{c} denotes of complement of c . Let Σ be a set of unabbreviated constraints and Ω be a set of constraint expressions.

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

- (i) For each concept expression 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 expression e , then there must be exactly one node in γ labeled with \bar{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 \bar{f} and \bar{e} , respectively.
- (vi) These are the only nodes and arcs of $g(\Sigma)$. □

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

If a node K of $G(\Sigma, \Omega)$ is labeled with an expression e , then \bar{K} denotes the node labeled with \bar{e} (which may be K itself). We use $K \rightarrow M$ to indicate that there is a path in $G(\Sigma, \Omega)$ from K to M .

Definition 4. Let $G(\Sigma, \Omega) = (\eta, \varepsilon, \lambda)$ be the labeled graph that represents 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 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 non-negated concept expression h such that M and N are labeled with h and $-h$, and $K \rightarrow M$ and $K \rightarrow N$.
- (ii) K is a \perp -node with level $n+1$ iff
 - a. There is a \perp -node M of level n , distinct from K , such that $K \rightarrow M$, and M is the \perp -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 \perp -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 \perp -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 labeled graph that represents Σ and Ω . Let K be a node of $G(\Sigma, \Omega)$. We say that K is a \perp -node iff K is a \perp -node with level n , for some non-negative integer n . We also say that K is a \top -node iff \bar{K} is a \perp -node. \square

Finally, we introduce the **IMPLIES** procedure (in Fig. 4) to test logical implication for extralite ontologies, whose soundness and completeness is established in [7].

```

IMPLIES( $\Sigma, e \sqsubseteq f$ )
input:    a set  $\Sigma$  of unabbreviated constraints and an unabbreviated constraint  $e \sqsubseteq f$ 
output:  “YES -  $\Sigma$  logically implies  $e \sqsubseteq f$ ”
           “NO -  $\Sigma$  does not logically imply  $e \sqsubseteq f$ ”

begin Construct  $G(\Sigma, \{e, f\})$ , the representation graph for  $\Sigma$  and  $\{e, f\}$ ;
if the node of  $G(\Sigma, \{e, f\})$  labeled with  $e$  is a  $\perp$ -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 -  $\Sigma$  logically implies  $e \sqsubseteq f$ ”;
else return “NO -  $\Sigma$  does not logically imply  $e \sqsubseteq f$ ”;
end

```

Fig. 4. The IMPLIES procedure

3.4 Examples of Extralite Ontologies

The following example illustrates the concepts introduced thus far, using those parts of the Music Ontology introduced in Section 2 (to save space, examples throughout the text use only some parts of the Music Ontology).

Example 1. Let $AGL = (V_{AGL}, C_{AGL})$ be the ontology that corresponds to the part of the Music Ontology shown in Fig. 2. Fig. 5 formalizes the set C_{AGL} of constraints and Fig. 6 depicts the graph $G(C_{AGL})$ that represents C_{AGL} (using unabbreviated constraints). Each constraint $e \sqsubseteq f$ in Fig. 5 corresponds to two arcs in Fig. 6: the arc from the node labeled with e to the node labeled with f , and the arc from the node labeled with \bar{f} to the node labeled with \bar{e} (where \bar{c} denotes of complement of c). Note that, in the constraint graph of Fig. 6, there is a path from $mo:Label$ to $\neg mo:SoloMusicArtist$, which indicate that C_{AGL} logically implies $mo:Label \sqsubseteq \neg mo:SoloMusicArtist$. That is, C_{AGL} logically implies that these two classes are disjoint. \square

Example 2. Let $SGL = (V_{SGL}, C_{SGL})$ be the ontology that corresponds to the $mo:Signal$ class, its subclasses and properties, shown on the bottom-left of Fig. 1. Fig. 7 formalizes the set of constraints C_{SGL} and Fig. 8 contains the graph $G(C_{SGL})$ that represents C_{SGL} (using unabbreviated constraints). $G(C_{SGL})$ is constructed as the graph in Fig. 6. In particular, note that there is a path in $G(C_{SGL})$ from $(\geq 2 mo:sampled_version)$ to

Constraint	Informal specification
$(\geq 1 \text{ mo:member_of}) \sqsubseteq \text{foaf:Person}$ $(\geq 1 \text{ mo:member_of}^-) \sqsubseteq \text{foaf:Group}$	The domain of mo:member_of is foaf:Person The range of mo:member_of is foaf:Group
$\text{mo:MusicArtist} \sqsubseteq \text{foaf:Agent}$ $\text{foaf:Group} \sqsubseteq \text{foaf:Agent}$ $\text{foaf:Organization} \sqsubseteq \text{foaf:Agent}$ $\text{mo:SoloMusicArtist} \sqsubseteq \text{foaf:Person}$ $\text{mo:SoloMusicArtist} \sqsubseteq \text{mo:MusicArtist}$ $\text{mo:MusicGroup} \sqsubseteq \text{mo:MusicArtist}$ $\text{mo:MusicGroup} \sqsubseteq \text{foaf:Group}$ $\text{mo:CorporateBody} \sqsubseteq \text{foaf:Organization}$ $\text{mo:Label} \sqsubseteq \text{mo:CorporateBody}$	mo:MusicArtist is a subset of foaf:Agent foaf:Group is a subset of foaf:Agent foaf:Organization is a subset of foaf:Agent $\text{mo:SoloMusicArtist}$ is a subset of foaf:Person $\text{mo:SoloMusicArtist}$ is a subset of mo:MusicArtist mo:MusicGroup is a subset of mo:MusicArtist mo:MusicGroup is a subset of foaf:Group mo:CorporateBody is a subset of foaf:Organization mo:Label is a subset of mo:CorporateBody
$\text{foaf:Person} \sqsubseteq \neg \text{foaf:Organization}$	foaf:Person and foaf:Organization are disjoint

Fig. 5. Constraints of *AGL* (unabbreviated form)

$\neg(\geq 2 \text{ mo:sampled_version})$. This means that C_{SGL} logically implies that $(\geq 2 \text{ mo:sampled_version}) \sqsubseteq \neg(\geq 2 \text{ mo:sampled_version})$ or, equivalently, C_{SGL} logically implies that $(\geq 2 \text{ mo:sampled_version}) \sqsubseteq \perp$. Intuitively, the set of individuals that $\text{mo:sampled_version}$ maps to two or more individuals is empty, that is, $\text{mo:sampled_version}$ is a functional property. Similar remarks apply to mo:isrc , implying that mo:isrc is an inverse functional property (i.e., a key). \square

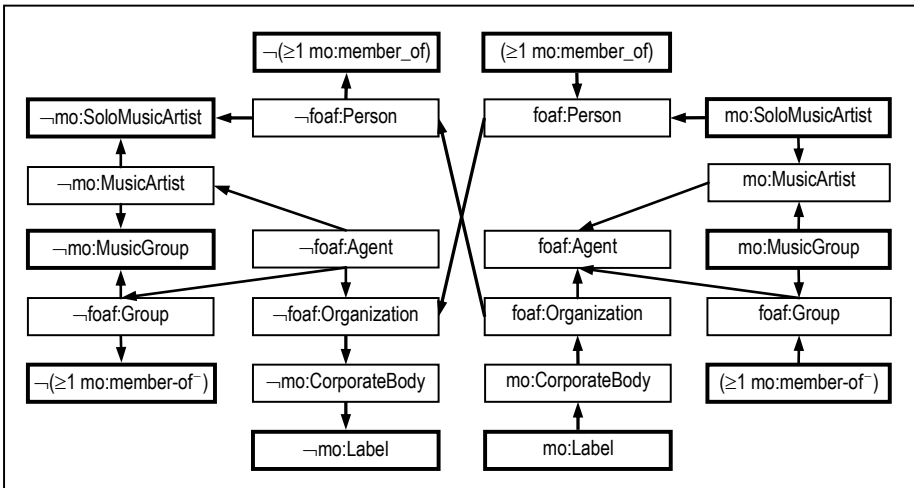


Fig. 6. The graph $G(C_{AGL})$ representing the constraints of *AGL*

Constraint	Informal specification
$(\geq 1 \text{ mo:sampled_version}) \sqsubseteq \text{mo:AnalogSignal}$ $(\geq 1 \text{ mo:sampled_version}^-) \sqsubseteq \text{mo:DigitalSignal}$ $(\geq 1 \text{ mo:isrc}) \sqsubseteq \text{mo:Signal}$ $(\geq 1 \text{ mo:isrc}^-) \sqsubseteq \text{xsd:String}$	The domain of $\text{mo:sampled_version}$ is mo:AnalogSignal The range of $\text{mo:sampled_version}$ is mo:DigitalSignal The domain of mo:isrc is mo:AnalogSignal The range of mo:isrc is mo:DigitalSignal
$\text{mo:AnalogSignal} \sqsubseteq \neg(\geq 2 \text{ mo:sampled_version})$ $\text{mo:String} \sqsubseteq \neg(\geq 2 \text{ mo:isrc}^-)$	$\text{mo:sampled_version}$ maps each individual in mo:AnalogSignal to at most one individual mo:isrc^- maps each individual in mo:String to at most one individual
$\text{mo:AnalogSignal} \sqsubseteq \text{mo:Signal}$ $\text{mo:DigitalSignal} \sqsubseteq \text{mo:Signal}$	mo:AnalogSignal is a subset of mo:Signal mo:DigitalSignal is a subset of mo:Signal
$\text{mo:DigitalSignal} \sqsubseteq \neg \text{mo:AnalogSignal}$	mo:DigitalSignal and mo:AnalogSignal are disjoint

Fig. 7. Constraints of *SGL* (unabbreviated form)

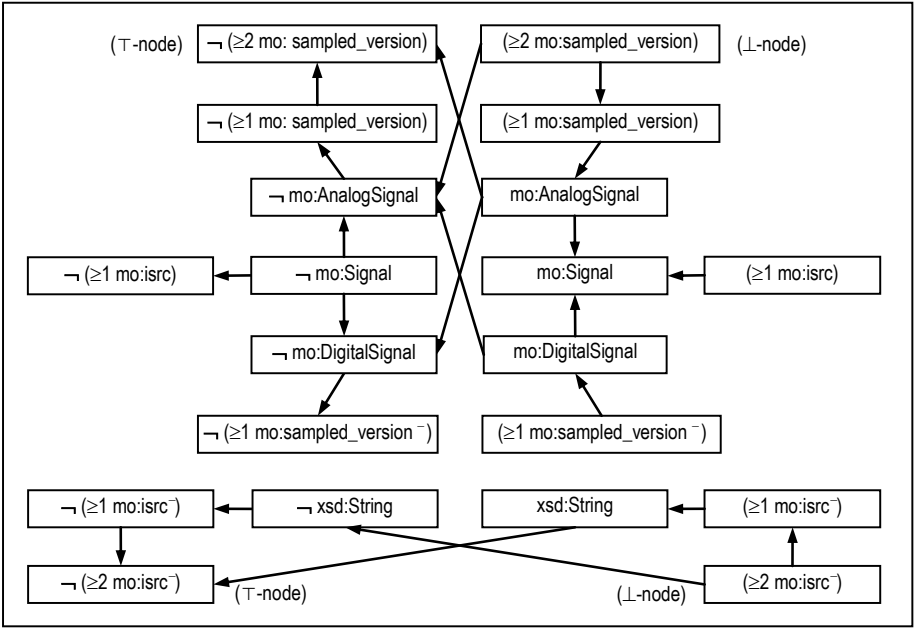


Fig. 8. The graph $G(C_{SGL})$ representing the constraints of *SGL*

4 Open Fragments of Domain Ontologies

Let $D = (V_D, C_D)$ denote the domain ontology, and $A = (V_A, C_A)$ denote the application ontology. The design of the application ontology A depends on what requirements it must satisfy, discussed in detail in this and the next sections.

Recall that C_D/V_A denotes the set of formulas σ using only symbols in V_A such that C_D logically implies σ . Consider the following set of requirements:

- R0. V_A is a subset of V_D
 R1. C_A logically implies C_D/V_A
 R2. Data exported by the data source satisfies C_A

An application ontology A that satisfies R0 and R1 is called an *open fragment* of D . Requirement R0 guarantees that the data is exported using a subset of the vocabulary of the domain ontology. Requirements R1 and R2 indicate that the data published by the data source will be consistent with all constraints that can be derived from C_D and that use only symbols in V_A . Intuitively, Requirements R0, R1 and R2 imply that any Web application that processes data modeled according to D and *that uses only the classes and properties in V_A* will also be able to process the data published by the data source (since the application expects data consistent with C_D/V_A).

Assume that the designer has already created V_A by selecting symbols from V_D so that R0 is trivially satisfied. Procedure **OpenFragment** (in Figure 9) generates C_A so that R1 is satisfied, based on the representation graph of C_D . The procedure does not guarantee, however, that $A = (V_A, C_A)$ is a strict extralite ontology since it does not try to generate missing domain and range constraints.

```

OpenFragment( $C_D, V_A; C_A$ )
:    the set  $C_D$  of normalized constraints of the domain ontology
           the vocabulary  $V_A$  of the application ontology
:   the set of constraints  $C_A$  of the application ontology
begin   Initialize  $C_A = \emptyset$ ;
           Construct  $G(C_D)$ , the representation graph for  $C_D$ ;
           Mark all nodes of  $G(C_D)$  labeled with expressions that use only
           atomic concepts and atomic roles in  $V_A$ ;
for each pair of nodes  $M$  and  $N$  of  $G(C_D)$ 
  if  $M$  and  $N$  are marked and there is a path from  $M$  to  $N$  in  $G(C_D)$ 
  then add  $e \sqsubseteq f$  to  $C_A$  where
     $e$  and  $f$  are expressions that label nodes  $M$  and  $N$ , respectively, and
     $e$  and  $f$  are expression of  $V_A$ , and
     $e \sqsubseteq f$  is an allowed constraint (in the sense of Section 2), and
     $\bar{f} \sqsubseteq \bar{e}$  is not already in  $C_A$  /* to avoid redundant constraints */
return  $C_A$ 
end
  
```

Fig. 9. Procedure OpenFragment

OpenFragment is an almost direct variation of **IMPLIES**, introduced at the end of Section 3.3. It generates all constraints that involve only symbols in V_A and that are logical consequences of C_D . However, it avoids generating both $e \sqsubseteq f$ and $\bar{f} \sqsubseteq \bar{e}$, which are equivalent. We note that **OpenFragment** is non-deterministic since the set of constraints generated depends on the order that the for-loop selects pairs of nodes of $G(C_D)$, which is not unique.

The above argument can be generalized into a correctness proof of the **Open-Fragment** procedure, in the following sense:

Theorem 1. Let C_D be the set of unabbreviated constraints of the domain ontology and V_A be the vocabulary of the application ontology. Let C_A be the set of constraints which **OpenFragment** outputs for C_D and V_A . Then, C_A logically implies C_D/V_A . \square

We close this section with an example that illustrates how the **OpenFragment** procedure operates.

Example 3. Assume that the domain ontology is $AGL = (V_{AGL}, C_{AGL})$, introduced in Example 1, and that the designer wants to formally specify the *Artist Contract* application ontology $AC = (V_{AC}, C_{AC})$, informally introduced in Section 2 as an open fragment of *MO*. He starts by defining the vocabulary V_{AC} by selecting symbols from V_{AGL} :

(1) $V_{AC} = \{ \text{mo:SoloMusicArtist, mo:MusicGroup, mo:label, mo:member_of} \}$

Then, **OpenFragment** generates the following set of constraints C_{AC} for AC :

- (2) $\text{mo:SoloMusicArtist} \sqsubseteq \neg \text{mo:Label}$
 (3) $\text{mo:Label} \sqsubseteq \neg(\geq 1 \text{ mo:member_of})$

Recall that Fig. 6 shows $G(C_{AGL})$, the representation graph for C_{AGL} . To help follow this example, the thicker boxes in Fig. 6 indicate the marked nodes (that contain terms in V_{AC}) and the thicker lines indicate the paths between marked nodes.

Indeed, **OpenFragment** outputs:

- the constraint in (2) since there is a path from $\text{mo:SoloMusicArtist}$ to $\neg \text{mo:Label}$, which implies that (2) is a logical consequence of C_{AGL}
- the constraint in (3) since there is a path from mo:Label to $\neg(\geq 1 \text{ mo:member_of})$, which implies that (3) is a logical consequence of C_{AGL} . Note that this constraint was not anticipated in the informal analysis at the end of Section 2 since it is not an immediate, trivial consequence of the constraints in C_{AGL}

In fact, constraints (2) and (3) use only symbols in V_{AC} and they are logical consequences of C_{AGL} (albeit not necessarily in C_{AGL}). They also meet Requirement R1 by Theorem 1.

However, **OpenFragment** will not output, for example, the following formulas:

- (4) $(\geq 1 \text{ mo:member_of}) \sqsubseteq \neg \text{mo:Label}$
 (5) $\text{mo:Label} \sqsubseteq \neg \text{mo:SoloMusicArtist}$

The procedure does not output (4) since this is not an allowed constraint, and it does not output (5) because (2) is already in C_{AGL} . However, since **OpenFragment** is non-deterministic, it could have returned (5) instead of (2). \square

5 Closed Fragments of Domain Ontologies

Let $D = (V_D, C_D)$ be the domain ontology, and $A = (V_A, C_A)$ the application ontology. Let C_A^+ be the set of constraints C_A extended with new axioms of the form $C \sqsubseteq \perp$ (or $(\geq 1 P) \sqsubseteq \perp$) that force each class C (or property P) in V_D , but not in V_A , to be the empty set. We now consider a different set of requirements:

- R0. V_A is a subset of V_D
 R1'. C_A^+ logically implies C_D
 R2'. Data exported by the data source satisfies C_A^+

An application ontology A that satisfies R0 and R1' is called a *closed fragment* of D . Requirement R0 again guarantees that the data is exported in a subset of the vocabulary of the domain ontology. Requirements R1' and R2' indicate that the data published by the data source satisfies C_D , when each class C (or property P) in V_D , but not in V_A , is taken to be the empty set.

ClosedFragment($C_D, V_A; V_A, C_A, S$)

input: the set C_D of normalized constraints of the domain ontology
 the vocabulary V_A of the application ontology

output: a new version of the vocabulary V_A of the application ontology
 the set of constraints C_A of the application ontology
 a set S of suggested mapping definitions

begin Initialize $C_A = \emptyset$ and $S = \emptyset$;
 Construct $G(C_D)$, the constraint graph for C_D ;
 /* Stage 1: Analyze nodes of $G(C_D)$ that contain expressions in V_A */
 Mark all nodes of $G(C_D)$ labeled with expressions that use only atomic concepts
 and atomic roles in V_A ;
 Create a new graph G_A by deleting any node N from $G(C_D)$ such that
 N is labeled with positive expressions and
 N has no antecedent which is marked and labeled with a positive expression, or
 N is labeled with negative expressions and
 N has no descendent which is marked and labeled with a negative expression;
 /* Stage 2: Generate definitions for the classes and properties to be added to V_A */
for each node N of G_A , in topological reverse order (i.e., from sinks to sources) **do**
 begin **if** N is labeled with a class E not in V_A
 then add E to V_A ;
 add " $E \equiv s_1 \sqcup \dots \sqcup s_n$ " to S , where E labels a node M and
 s_1, \dots, s_n label nodes M_1, \dots, M_n , and
 M_1, \dots, M_n are all nodes such that (M_k, M) is in G_A ;
 if N is labeled with an expression involving a property P not in V_A
 then add P to V_A ;
 add "*Skolemize*[P, G_A]" to S ; /* (see explanation in the text) */
 end
 /* Stage 3: Generate the constraints of the application ontology */
for each arc (M, N) of G_A
 add $e \sqsubseteq f$ to C_A where
 e and f are expressions that label nodes M and N , respectively, and
 e and f are expression of V_A , and
 $e \sqsubseteq f$ is an allowed constraint (in the sense of Section 2), and
 $\bar{f} \sqsubseteq \bar{e}$ is not already in C_A ; /* to avoid redundant constraints */
return V_A, C_A, S
end

Fig. 10. Procedure ClosedFragment

Intuitively, Requirements R0, R1' and R2' imply that any Web application that processes data modeled according to D will also be able to process data published by the data source, when each class C (or property P) in V_D , but not in V_A , is taken to be the empty set.

Assume that the designer has already created V_A by selecting symbols from V_D so that R0 is trivially satisfied. Procedure **ClosedFragment** (in Figure 10) extends V_A and creates C_A so that R1' is satisfied. Unlike **OpenFragment**, it guarantees that $A=(V_A, C_A)$ is a strict extralite ontology since it generates domain and range constraints for all properties in V_A .

ClosedFragment has three stages. The first stage is preparatory for the next stages and analyzes which nodes of the constraint graph of C_D have expressions using only symbols in V_A .

The second stage includes a class E in V_A , if E is in V_D , but not in V_A , and there is an expression s_i using only symbols in V_A which the constraints in C_D force to be a non-empty subset of E . If this is the case, E cannot be forced to be always empty (by an axiom of the form $E \sqsubseteq \perp$). The solution is to include E in V_A and define E as the union of all expression s_i using only symbols in V_A such that there is a constraint of the form $s_i \sqsubseteq E$ which is a logical consequence of C_D . Note that, to be correct, this stage has to process nodes in topological reverse order.

The second stage also adds properties to V_A , if necessary. Let P be a property in V_D , but not in V_A . Assume that there is an expression s_i using only symbols in V_A which the constraints in C_D force to be a non-empty subset of an expression p involving property P . If this is the case, P cannot be forced to be always empty (by an axiom of the form $(\geq 1 P) \sqsubseteq \perp$). The solution is to include P in V_A and define P as follows. Let s be the union of all expression s_i using only symbols in V_A such that there is a constraint of the form $s_i \sqsubseteq p$ which is a logical consequence of C_D . Then, P is defined as the set of all pairs of individuals (x,y) such that x is in the set denoted by s and y is one or more new individuals introduced until p is satisfied. We denote by *Skolemization* $[P, G_A]$ such definition for P , and note that it is not expressible in the attributive languages of Section 3.1. The details of this construction and why it is always possible is outside the scope of this paper.

The third stage of **ClosedFragment** is a variation of **IMPLIES**. However, we note that this stage also automatically generates missing domain and range constraints. Indeed, assume that a property P is in V_A and that $(\geq 1 P) \sqsubseteq E$ is the domain constraint for P in C_D . Then, the constraint graph for C_D will have an arc from the node labeled with $(\geq 1 P)$ to the node labeled with E . Since P is in V_A , the first stage will then add E to V_A , if E is not already in V_A . Then, the third stage will add $(\geq 1 P) \sqsubseteq E$ to C_A . An entirely similar argument applies to range constraints.

The above argument can be generalized into a correctness proof of the **Closed-Fragment** procedure, in the following sense:

Theorem 2. Let C_D be the set of unabbreviated constraints of the domain ontology and V_A be the original vocabulary of the application ontology. Let \bar{V}_A and C_A be the vocabulary and the set of constraints that **ClosedFragment** outputs for C_D and V_A . Then, C_A^+ logically implies C_D , where C_A^+ is defined with respect to \bar{V}_A . \square

We conclude this section with an example that illustrates how the **ClosedFragment** procedure operates.

Example 4. Let $ME = (V_{ME}, C_{ME})$ be an ontology that formalizes the class hierarchies of the Music Ontology rooted at classes `event:Event` and `frbr:Expression`, informally introduced in Fig. 1 of Section 2. The reader may verify that $SGL = (V_{SGL}, C_{SGL})$, the Signal ontology defined in Example 2, is an open fragment of ME . The goal in this example is to redefine SGL so that it becomes a closed fragment of ME , using the **ClosedFragment** procedure.

Recall from Example 2 that V_{SGL} is:

- (1) $V_{SGL} = \{ \text{mo:DigitalSignal}, \text{mo:analogSignal}, \text{mo:Signal}, \text{xsd:String}, \text{mo:sampled_version}, \text{mo:isrc} \}$

Let \bar{V}_{SGL} be the new vocabulary, \bar{C}_{SGL} be the set of constraints and S be the set of mapping definitions that **ClosedFragment** outputs, when given C_{ME} and V_{SGL} as input. To construct \bar{V}_{SGL} , \bar{C}_{SGL} and S , **ClosedFragment** uses the constraint graph of C_{ME} , not shown here to save space (but the reader may observe Fig. 1 for an informal description of C_{ME}).

Then, \bar{V}_{SGL} is the set:

- (2) $\bar{V}_{SGL} = \{ \text{mo:DigitalSignal}, \text{mo:analogSignal}, \text{mo:Signal}, \text{xsd:String}, \text{mo:sampled_version}, \text{mo:isrc}, \text{mo:MusicalExpression}, \text{mo:Expression} \}$

\bar{C}_{SGL} is shown in Fig. 11. Note that it includes two constraints not in C_{SGL} :

- (3) $\text{mo:Signal} \sqsubseteq \text{mo:MusicalExpression}$
- (4) $\text{mo:MusicalExpression} \sqsubseteq \text{mo:Expression}$

S contains the mapping definitions:

- (5) $\text{mo:MusicalExpression} \equiv \text{mo:Signal}$
- (6) $\text{mo:Expression} \equiv \text{mo:MusicalExpression}$

In particular, we observe that the mapping definitions in (5) and (6) force classes `mo:MusicalExpression` and `mo:Expression` to have the same set of individuals as `mo:Signal`. The definitions in (5) and (6) indeed indicate that the data exported will trivially satisfy the constraints in (3) and (4).

Finally, we note that, by definition of closed fragment, all classes and properties in V_{ME} , but not in \bar{V}_{SGL} , will be empty. □

$(\geq 1 \text{ mo:sampled_version}) \sqsubseteq \text{mo:AnalogSignal}$ $(\geq 1 \text{ mo:sampled_version}^-) \sqsubseteq \text{mo:DigitalSignal}$ $(\geq 1 \text{ mo:isrc}) \sqsubseteq \text{mo:Signal}$ $(\geq 1 \text{ mo:isrc}^-) \sqsubseteq \text{xsd:String}$	$\text{mo:String} \sqsubseteq \neg (\geq 2 \text{ mo:isrc}^-)$ $\text{mo:AnalogSignal} \sqsubseteq \neg (\geq 2 \text{ mo:sampled_version})$	$\text{mo:AnalogSignal} \sqsubseteq \text{mo:Signal}$ $\text{mo:DigitalSignal} \sqsubseteq \text{mo:Signal}$ $\text{mo:DigitalSignal} \sqsubseteq \neg \text{mo:AnalogSignal}$ $\text{mo:Signal} \sqsubseteq \text{mo:MusicalExpression}$ $\text{mo:MusicalExpression} \sqsubseteq \text{mo:Expression}$
--	--	--

Fig. 11. Constraints of SGL as a closed fragment of ME (unabbreviated form)

6 Related Work

The results reported in the paper cover a topic – the role that constraints play in the design of Linked Data – that is much neglected in the literature. The question of Linked Data semantics is not new, though. Recent investigation [11][12][15] 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 [10]. Indeed, RDB-to-RDF tools (see [18] 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. We introduced the notions of open and closed ontology fragments exactly to address this question. Such notions have no parallel in the published literature.

We note that the problem we address cannot be reduced to a question of ontology alignment in the context of Linked Data, addressed for example in [16][21]. Indeed, we stress that the problem we focus on refers to bootstrapping an application ontology (including constraints) as a fragment of a domain ontology.

The results in the paper also contribute to improving ontology browsing tools based on the idea of *focus+context* [20], where the notion of focus would be carried out by a vocabulary selection and the notion of context would be provided by the constraints. The methods to construct fragments of the domain ontology would act as a lens through which the user would browse the (large) domain ontology.

7 Conclusions

In this paper, we introduced automatic methods for constructing application ontology constraints, when the application ontology is an open or a closed fragment of the domain ontology. The final set of constraints will have useful properties, as detailed in Sections 4 and 5. The methods assume that the ontologies are written in an expressive family of attributive languages and depend on a procedure to test logical implication, based on constraint graphs.

The results in the paper are directly mapped to the RDF context and cover a topic – the role that constraints play in the design of Linked Data – that is much neglected in the literature.

As for current work, we are modifying an RDB-to-RDF tool [19] to generate application ontology constraints, as described in the paper. We are also extending the strategy to account for complex source-to-ontology mappings, using results from [13], to other types of constraints, using the development reported in [9].

Acknowledgements. This work was partly supported by CNPq, under grants 473110/2008-3 and 557128/2009-9, by FAPERJ under grant E-26/170028/2008, and by CAPES under grant NF 21/2009.

References

- [1] Baader, F., Nutt, W.: Basic Description Logics. In: Baader, F., Calvanese, D., McGuinness, 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)
- [2] Berners-Lee, T.: *Linked Data - Design Issues* (2006), <http://www.w3.org/DesignIssues/LinkedData.html> (retrieved July 23, 2006)
- [3] Berrueta, D., Phipps, J.: Best Practice Recipes for Publishing RDF Vocabularies - W3C Working Group Note, <http://www.w3.org/TR/swbp-vocab-pub/> (accessed June 14, 2009)
- [4] Bizer, C., Cyganiak, R., Heath, T.: How to publish Linked Data on the Web, <http://www4.wiwiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/> (accessed June 14, 2009)
- [5] 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)
- [6] Brickley, D., Miller, L. FOAF Vocabulary Specification 0.98. Namespace Document August 9, 2010 - Marco Polo edn., latest version [http://xmlns.com/foaf/spec/\(rdf,wiki\)](http://xmlns.com/foaf/spec/(rdf,wiki))
- [7] 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, 1274–1301 (2010)
- [8] Casanova, M.A., Breitman, K.K., Furtado, A.L., Vidal, V.M.P., Macêdo, J.A.F. The Role of Constraints in Linked Data. MCC21/11, Dept. Informatics, PUC-Rio (April 2011)
- [9] 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* (to appear)
- [10] Das, S., Sundara, S., Cyganiak, R.: R2rml: RDB to RDF mapping language. W3C RDB2RDF working group, <http://www.w3.org/TR/r2rml/> (accessed December 15, 2010)
- [11] 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)
- [12] 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)
- [13] Lauschner, T., Casanova, M.A., Vidal, V.M.P., Macedo, J.A.F.: Efficient Decision Procedures for Query Containment and Related Problems. In: *Proc. XXIV Brazilian Symposium on Databases* (2009)
- [14] Madison, O.: (Chair) *Functional Requirements for Bibliographic Records - Final Report*. IFLA Study Group on the Functional Requirements for Bibliographic Records (February 2009), <http://www.ifla.org/VII/s13/frbr/>
- [15] McCusker, J., McGuinness, D.L.: owl:sameAs considered harmful to provenance. In: *Proc. ISCB Conference on Semantics in Healthcare and Life Sciences* (2010)
- [16] 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)

- [17] Raimond, Y., Giasson, F.: Music Ontology Specification. Specification Document (November 28, 2010), latest version
[http://purl.org/ontology/mo/\(RDF/XML, Turtle\)](http://purl.org/ontology/mo/(RDF/XML,Turtle))
- [18] 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)
- [19] Salas, P.E., Breitman, K.K., Viterbo, J., Casanova, M.A.: Interoperability by Design Using the Std-Trip Tool: an a priori approach. In: Proc. 6th Int'l. Conf. on Semantic Systems (I-SEMANTICS 2010), Graz (2010)
- [20] Villegas, A., Olivé, A.: A Method for Filtering Large Conceptual Schemas. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 247–260. Springer, Heidelberg (2010)
- [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)