

# On the Implementation of an Algebra of Lightweight Ontologies

Rômulo C. Magalhães  
*Departamento de Informática,  
PUC-Rio,  
Rio de Janeiro, Brasil*  
[rmagalhaes@inf.puc-rio.br](mailto:rmagalhaes@inf.puc-rio.br)

Marco A. Casanova  
*Departamento de Informática,  
PUC-Rio,  
Rio de Janeiro, Brasil*  
[casanova@inf.puc-rio.br](mailto:casanova@inf.puc-rio.br)

Bernardo P. Nunes  
*PUC-Rio,  
UNIRIO  
Rio de Janeiro, Brasil*  
[bernardo@ccead.puc-rio.br](mailto:bernardo@ccead.puc-rio.br)

Giseli Rabello Lopes  
*Departamento de Ciência da  
Computação,  
UFRJ,  
Rio de Janeiro, Brasil*  
[giseli@dcc.ufrj.br](mailto:giseli@dcc.ufrj.br)

## ABSTRACT

This paper<sup>1</sup> first argues that ontology design may benefit from treating ontologies as theories and from the definition of a set of operations that map ontologies into ontologies, especially their constraints. The paper then defines the class of ontologies used and proposes four operations to manipulate them. It proceeds to discuss how the operations may help design new ontologies. The core of the paper describes an implementation of the operations as a Protégé plug-in, called *OntologyManagerTab*, and includes use case examples to validate the discussion.

## CCS CONCEPTS

**Data Design, Evolution and Migration; Data Integration; Data Models; Semantic Web and Databases.**

## KEYWORDS

Constraint Definition; Ontology Design; Linked Data; Protégé; *OntologyManagerTab*.

## 1 INTRODUCTION

The Linked Data principles [4][5] suggest a way to publish databases on the Web that facilitates interoperability. These principles emphasize the definition of the conceptual structure of the data through the reuse of known ontologies, thus minimizing the need for alignment between conceptual schemas, a difficult and error prone task, which lies at the core of the interoperability issue. We argued elsewhere that certain familiar

ontology design problems are profitably addressed by treating ontologies as theories and by defining a set of operations on ontologies [7][9].

Briefly, we define an ontology as a pair  $O=(V, \Sigma)$  such that  $V$  is a vocabulary and  $\Sigma$  is a set of constraints in  $V$ . The theory of  $\Sigma$  is the set of all constraints that are logical consequences of  $\Sigma$ . We emphasize that the constraints in  $\Sigma$  capture the semantics of the terms in  $V$  and must, therefore, be brought to the foreground. The theory of  $\Sigma$  identifies the constraints that are implicitly defined, but which must be considered when using the ontology. The operations we propose create new ontologies out of other ontologies taking into account their constraints, and facilitate the domain specialist to uphold the Linked Data principles. Such operations extend the idea of namespaces to consider constraints and help address familiar ontology design problems, which we now outline.

Consider first the problem of designing an ontology to publish data on the Web. If the designer follows the Linked Data principles, he must select known ontologies, as much as possible, to organize the data so that applications “can dereference the URIs that identify vocabulary terms in order to find their definition”. 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. Furthermore, he should publish the data so that the original semantics of the terms is preserved. To facilitate ontology design from this perspective, we introduce three operations on ontologies, called projection, union and deprecation.

Consider now the problem of comparing the expressive power of two ontologies,  $O_1=(V_1,\Sigma_1)$  and  $O_2=(V_2,\Sigma_2)$ . If the designer wants to know what they have in common, he should create a mapping between their vocabularies and detect which constraints hold in both ontologies, after the terms are appropriately mapped. The intersection operation answers this question. We argued elsewhere [6] that intersection is also useful to address the design of mediated schemas that combine export schemas in a way that the data exposed by the mediator is always consistent.

On the other hand, if the designer wants to know what holds in  $O_1=(V_1,\Sigma_1)$ , but not in  $O_2=(V_2,\Sigma_2)$ , he should again create a mapping between their vocabularies and detect which constraints hold in the theory of  $\Sigma_1$ , but not in the theory of  $\Sigma_2$ ,

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [Permissions@acm.org](mailto:Permissions@acm.org).

IDEAS '17, July 12–14, 2017, Bristol, United Kingdom

© 2017 Association for Computing Machinery.

ACM ISBN 978-1-4503-5220-8/17/07...\$15.00

<http://dx.doi.org/10.1145/3105831.3105856>

after the terms are appropriately mapped. The difference operation answers this question. Likewise, if the user wants to analyze what changed from one version of an ontology to the other, he should also use the difference operation.

The core of this paper describes an implementation of projection, union, intersection and difference as a Protégé plugin, called *OntologyManagerTab*, to assist the domain specialist in the process of managing ontologies for data publication, it includes use case examples to validate the discussion. The machinery to handle constraints developed in [6][7][8] and [9] provides the theoretical foundations of the paper. Previous work by the authors [7] introduced the notion of open fragment, which is captured by projection. The design of mediated schemas was addressed in [6]. A preliminary implementation of the operations was described in [16].

The paper is organized as follows. Section 2 reviews background concepts. Section 3 introduces the operations. Sections 4 summarizes related work. Section 5 and 6 discuss the implementation of the operations and present examples. Section 7 contains the conclusions and future work.

## 2 BACKGROUND

We adopt the basic notion of Description Logic [3]. Very briefly, a *vocabulary*  $V$  consists of a set of *atomic concepts*, a set of *atomic roles*, and the *bottom concept*  $\perp$ . A *language* in  $V$  is a set of strings, using symbols in  $V$ , defining the set of *concept descriptions* in  $V$  and the set of *role descriptions* in  $V$ . An *inclusion* in  $V$  is a string of the form  $u \sqsubseteq v$ , where  $u$  and  $v$  both are concept descriptions in  $V$  or both are role descriptions in  $V$ . Table 1 shows common types of inclusions.

An *ontology* is a pair  $O=(V,\Sigma)$  such that  $V$  is a vocabulary and  $\Sigma$  is a set of inclusions in  $V$ , called the *ontology constraints*. The *theory* of  $\Sigma$  (or the *theory* of  $O$ ), denoted  $\tau[\Sigma]$  (or  $O[\Sigma]$ ), is the set of all logical consequences of  $\Sigma$ .

We say that two sets of inclusions,  $\Gamma$  and  $\Theta$ , are *equivalent*, denoted  $\Gamma \equiv \Theta$ , iff their theories are equal, that is, the set of all logical consequences of  $\Gamma$  is equal to that of  $\Theta$ . Likewise, two ontologies  $O_1=(V_1,\Sigma_1)$  and  $O_2=(V_2,\Sigma_2)$  are *equivalent*, also denoted  $O_1 \equiv O_2$ , iff  $\Sigma_1$  and  $\Sigma_2$  are equivalent.

We adopt in this paper the family of *lightweight languages*, which is equivalent to the family *DL - Lite<sub>core</sub><sup>N</sup>* [2] and the proof procedure based on constraint graphs developed in [6][7][8] and [9]. Table 1 lists the constraints allowed in the lightweight family.

## 3 OPERATIONS ON ONTOLOGIES

Definition 1 introduces the ontology operations we propose. It is important to highlight that the new ontology, obtained from the execution of these operations, presents a set of constraints that considers the semantics of the constraints of the ontologies involved.

**Table 1. Common inclusion types used in conceptual modeling.**

Name	Inclusion type	Informal semantics
Domain Constraint	$(\geq 1 P) \sqsubseteq C$	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	$(\geq 1 P^-) \sqsubseteq C$	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 $C \sqsubseteq (\geq k P^-)$	property $P$ or its inverse $P^-$ maps each individual in class $C$ to at least $k$ distinct individuals
maxCardinality Constraint	$C \sqsubseteq \neg(\geq k+1 P)$ 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	$C \sqsubseteq D$	each individual in $C$ is also in $D$ , that is, class $C$ denotes a subset of class $D$
Disjointness Constraint	$C \sqsubseteq \neg D$	no individual is in both $C$ and $D$ , that is, classes $C$ and $D$ are disjoint

**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  $\Psi$  be a set of constraints in  $V_1$ .

1. 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  $\Sigma_P$  is the set of constraints in  $\tau[\Sigma_1]$  that use only classes and properties in  $W$ .
2. The *deprecation* of  $\Psi$  from  $O_1=(V_1,\Sigma_1)$ , denoted  $\delta[\Psi](O_1)$ , returns the ontology  $O_D=(V_D,\Sigma_D)$ , where  $V_D=V_1$  and  $\Sigma_D=\Sigma_1-\Psi$ .
3. 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  $\Sigma_U=\Sigma_1 \cup \Sigma_2$ .
4. 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  $\Sigma_N=\tau[\Sigma_1] \cap \tau[\Sigma_2]$ .
5. The *difference* of  $O_1=(V_1,\Sigma_1)$  and  $O_2=(V_2,\Sigma_2)$ , denoted  $O_1 - O_2$ , returns the ontology  $O_F=(V_F,\Sigma_F)$ , where  $V_F=V_1$  and  $\Sigma_F=\tau[\Sigma_1] - \tau[\Sigma_2]$ .

Note that deprecation does not reduce to difference since, in general, we have

$$\tau[\Sigma_D] = \tau[\Sigma_1 - \Psi] \neq \tau[\Sigma_1] - \tau[\Psi]$$

We also note that the ontology  $O_R$  that results from an operation is unique, by definition. However, there might be several ontologies that are equivalent to  $O_R$ . For example, if  $O_P=(V_P,\Sigma_P)$  is the projection of  $O_1$  on  $W$ , there might be several sets of constraints that are equivalent to the set of constraints in the theory of  $O_1$  that use only terms in  $W$ .

The projection operation allows the designer to define a set  $W$  containing just a few terms from the vocabulary of an ontology. This set retains the semantics of the terms from the original vocabulary in  $W$  through the constraints derived from those of the original ontology. It is also the only operation among those implemented in this dissertation that has a single ontology as argument. The main advantage of the *Projection* procedure is to automate the onerous task of the domain specialist in the

formalization of new ontologies by extracting the needed concepts and their dependencies, allowing the reuse of widely consolidated terms with little work.

The applications that adopt the principles of Linked Data have the challenge of providing its users with integrated information from multiple data sources that may or may not contain overlapping data, hence the importance of the *Union* and *Intersection* operations.

Usually, the process of integrating multiple ontologies consists in the union of two versions of ontologies unbeknownst whether there was a common ontology that originated them. Also, the union between data from distinct domains may generate conflicts and inconsistencies depending on the versions of the ontologies used by each domain.

Similarly to the *Union* operation, the domain specialist may want to extract only the overlapping information while consulting multiple data sources. This is achieved by the *Intersection* operation.

Ontologies evolve over time due to changes in the domain they represent or due to the fact that they have been built in a collaborative way and, therefore, need to be updated to represent a common understanding to different users. To detect modifications between two versions of the same ontology, we have the *Difference* operation, that compares two ontologies and returns the terms and constraints that are present in the first, but not in the second.

According to (M. Klein, 2004), the differences between two ontologies can be classified as simple or complex. The simple differences are those that do not affect the structure of the ontology, such as the change of names of classes, properties or data types. On the other hand, complex differences are those that affect the ontology structure, include modifications in the class hierarchy or in constraints, such as disjunction and cardinality restrictions. The *Difference* operation addresses both types of differences.

## 4 RELATED WORK

Reference [11] 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 [17], [21], [10] and [15]. 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 over one or more existing ontologies.

Some tools, such as Prompt [14] and ODEMerge [18], allow the user to combine two or more ontologies in a semiautomatic or in

an automatic way. Other tools, such as PromptDiff [13] and OntoDiff [19], deal with ontology change detection. Our tool offers a complete environment to design and maintain ontologies, which allows applying a series of operations over one or more ontologies and enabling reuse, versioning, evolution and integration of ontologies.

Reference [20] proposes a tool that implements the projection operation by the creation of a database view resulting from query execution. However, this tool does not allow the generation of semantic information captured by the constraints that apply to the vocabulary terms.

Other tools, such as OAPT, proposed in [1], address the question of partitioning an ontology into modules to facilitate ontology reuse. The modularization operation is akin to our projection operation. However, we go further and consider other operations that, in the end, show how to combine modules into larger ontologies. Finally, previous work by the authors [7] introduced the notion of *open fragment*, captured by the projection operation, and [9] covered some of the operations discussed in this paper.

## 5 IMPLEMENTATION OF THE OPERATIONS

The *OntologyManagerTab* offers the ontology operations described in Section 3, integrated with traditional ontology management features. *OntologyManagerTab* was developed in Java as a tab plug-in of Protégé 3.4.8 (this implementation might require minor modifications to work with other versions of Protégé). However, *OntologyManagerTab* works in a completely independent manner from the main framework, using Protégé only as a Graphical User Interface (GUI) enclosure. In other words, all the functionalities provided by *OntologyManagerTab* do not rely on any of the Protégé libraries, making the tool easily adaptable as a plug-in for any other framework or as a stand-alone software.

*OntologyManagerTab* performs a procedure that loads only the *lightweight kernel* of an ontology, that is, the *lightweight constraints* of the ontology, according to the  $DL-Lite_{core}^N$  family. Thus, it guarantees that the operations will be executed only over *lightweight constraints*, consistently with the use of *constraint graphs*.

The software source code is divided into three packages: *Application*, *Ontology* and *Main*. The *Application* package contains classes that implement the Java interfaces required for the application to run. The *Main* package contains the *OntologyManagerTab* class, that implements the discussed operations and integrates the software with the Protégé framework. Finally, the *Ontology* package contains the classes that implement *constraint graphs*, as well as the procedures to load the *lightweight kernel* of ontologies and save the results obtained.

## 6 EXAMPLES

**Table 2. Experiments Processing Times in seconds.**

Sample	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	Average
<b>Parliament Ontology</b>											
Load	0,724	1.261	1.130	0.941	0.764	0.984	0.689	0.436	0.589	0.779	0.829
<b>Relationship Ontology</b>											
Load	0.955	0.728	0.649	0.883	0.823	0.792	0.589	0.743	1.219	0.509	0.789
<b>Parliament and Relationship Ontologies</b>											
Union	0.428	0.473	0.433	0.479	0.601	0.416	0.488	0.506	0.512	0.476	0.481
Intersection	0.402	0.483	0.488	0.530	0.471	0.421	0.521	0.490	0.472	0.497	0.477

This section summarizes experiments with the operations. We refer the reader to [12] for detailed experiments with full ontologies from several domains, as well as consistent fragments of these ontologies and a few ontologies created from scratch to test the base cases. All experiments were performed in a machine with a 2 GHz Intel Core 2 Duo processor, 8 GB 1067 MHz DDR3 of memory and using the Java Virtual Machine default configuration.

Table 2 at the end of the paper illustrates the projection operation. The upper left table in Table 2 shows a representation of the input ontology  $O_I$ , where each lightweight term used in a constraint of  $O_I$  appears at least once in the first column of the table and each constraint  $e \sqsubseteq f$  of  $O_I$  appears in a separate row, with  $e$  in column 1 and  $f$  in column 2. The upper right table allows the user to select the set of symbols  $W$  of the vocabulary to define the projection. The lower left table in Table 2 shows the results of the projection  $O_P=(V_P,\Sigma_P)$ , whereas in the first table each constraint  $e \sqsubseteq f$  appears in a separate row, with  $e$  in column 1 and  $f$  in column 2, as well as its inverse  $\bar{f} \sqsubseteq \bar{e}$ . For example, both  $Person \sqsubseteq \neg Organization$  and  $Organization \sqsubseteq \neg Person$  are shown in the lower left table; it is up to the user to decide which constraint to keep, since they are equivalent.

To execute the *projection* operation, the user first selects “Projection” on the dropdown list that specifies which operation will be executed. Next, the user loads the ontology from which he intends to extract the lightweight kernel by clicking the “Load Ontology Projection” button and selecting the base ontology file. At this point, the base ontology will be displayed in the table on the upper left and its classes and properties will be displayed in the table on the upper right. From this last table, the user then selects the terms to be used for the projection and clicks the “Run” button next to the operation dropdown list. The result of the projection will be displayed in the “Resulting Ontology” table on the lower left. To execute chained procedures, such as  $Union(Intersection(O_1,O_2),O_3)$ , the user must save the ontology resulting from the first operation,  $O_4 = Intersection(O_1,O_2)$ , and reload it to execute the second operation,  $Union(O_4,O_3)$ .

To better illustrate the usefulness of the application, consider the following **Scenario 1**: a domain specialist has created an ontology in which he defined certain terms and constraints that he could not find in any related ontology; later on, he discovered a well know ontology that contained some of the needed terms.

To exemplify **Scenario 1**, we will use the Parliament Ontology, created to specify the organization of the Brazilian government

as well as the relationship between politicians and members of the executive branch, as the first ontology created by the domain specialist. Its latest version can be obtained at [22] and a version with the latest data regarding the Brazilian 55<sup>th</sup> legislature of 2015 at [23]. We adopt the Relationship ontology, which contains a vocabulary for describing relationships between people, as the second ontology. Its latest version can be found at [24].

Using the *OntologyManagerTab*, the user can specify the type of operation that will be performed. When any operation, except *Projection*, is selected, the upper right table changes into a copy of the upper left table as shown Table 2. Also, when selecting these other operations, a checkbox and a spinner appear above the upper right table, giving the user the option of using word approximation. This can be done by selecting the checkbox and specifying the number of different characters to be accepted in the comparison; it will ignore blank spaces as well as disregard capitalization of letters and only count the different characters until the specified threshold. The recommended value for the word approximation threshold is 1, just to consider plurals and typos committed by the ontology designer. If the specified threshold is too big, the program may find matches that do not describe the same term, especially if the ontologies in question have classes or properties with small names.

Considering **Scenario 1**, the user can execute the *Intersection* between the Parliament and Relationship ontologies using the word approximation option to make sure there is a correlation between them, that is, if their intersection is not empty. After confirming their correlation, it is possible to use the *Union* between them, as shown in Table 2 to aggregate the rest of the relationships declared in the Relationship ontology to the Parliament ontology and save the resulting new ontology.

Another possibility would be that, after confirming the correlation between the two ontologies, the domain specialist would like to add only a few extra properties and classes from the Relationship ontology. To achieve this, the user would need to load the Relationship ontology and run a *Projection* over it to select the desired items, remembering to also include the classes and properties that are common between both ontologies, save the resulting ontology and afterwards run a *Union* between this saved ontology and the Parliament ontology, using the word approximation option with the same configuration as that used in the intersection.

Table 2 shows 10 processing time samples obtained for the operations of *Loading*, *Intersection* and *Union* executed in **Scenario 1** with word approximation threshold equal to 1. All measurements are in seconds and the last column shows the average. Still regarding processing times, the projections of approximately 15 or less classes and properties is almost instantaneous with an order of magnitude of 0.02 seconds. All these processing times were measured adding timestamps to the original code.

The only applications similar enough to ours, for comparison, would be Prompt [14] and ODEMerge [18], which, as said in Section 4 allow the user to combine two or more ontologies.

However, we were not able to add the same timestamps in order to have the correct metric for time correlation between them and **OntologyManagerTab**. That aside, on average they took close to 1.5 seconds to do the automatic part of the merge, which would be our *Union*.

As for the *Difference* operation provided by **OntologyManagerTab**, one may use it to compare versions of the same ontology to identify the updates made. This operation can also be used as an intermediate step at the domain specialist discretion. For instance, to perform *Deprecations*, the user may first perform a projection of the terms he wants to deprecate from the ontology and save the resulting ontology; afterwards he may run the *Difference* between the first ontology and the saved ontology. The processing times of the *Difference* operation are similar to those of the *Intersection* operation, and will vary according to the involved ontologies.

## 7 CONCLUSION AND FUTURE WORK

Few tools assist the domain specialist in the development of a new ontology that represents a correct understanding of the semantics of the ontologies involved. To address this issue, it is necessary to take into account not only the terms of the original ontologies but also their constraints. This is possible by considering ontologies as logical theories, composed of vocabularies and constraints, and defining the algebraic operations over them.

In this paper, we argued that ontology design may benefit from treating ontologies as theories and from the definition of a set of operations that map ontologies into ontologies, especially their constraints. The core of the paper outlined an implementation of the operations as a Protégé plug-in, the **OntologyManagerTab**, and included some examples to illustrate the discussion. We refer the reader to [12] for further examples.

As for future work, we intend to extend the tool to more expressive classes of ontologies, using the results in [8] and to improve the GUI with complementary visual representation.

**Acknowledgments.** This work was partly funded by CNPq, under grants 442338/2014-7 and 303332/2013-1, and by FAPERJ, under grant E-26/201.337/2014.

## REFERENCES

- [1] Algergawy, A., Babalou, S., Klan, F., Koenig-Ries, B., "OAPT: A tool for ontology analysis and partitioning," in Proc. of the 19th International Conference on Extending Database Technology, EDBT 2016.
- [2] Artale, A., Calvanese, D., Kontchakov, R., Zakharyashev, 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: F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider (eds), The Description Logic Handbook: Theory, Implementation and Applications, Cambridge U. Press, Cambridge, UK, 43–95, 2003.
- [4] Berners-Lee, T., "Linked Data - Design Issues," in W3C, 2006.
- [5] Bizer, C., Cyganiak, R., Heath, T., "How to publish Linked Data on the Web," in <http://www4.wiwi.fu-berlin.de/bizer/pub/LinkedDataTutorial/>, 2007.
- [6] 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" in Data & Knowledge Engineering 69(12), 1274–1301, 2010.
- [7] Casanova, M.A., Breitman, K.K., Furtado, A.L., Vidal, V.M.P., Macêdo, J.A.F., "The Role of Constraints in Linked Data," in Proceedings of the Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011, Part II. Lecture Notes in Computer Science v.7045. Springer, 781–799, 2011.
- [8] 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: Michael G. Hinchey (ed.), Conquering Complexity, Springer, 431–461, 2012a.
- [9] Casanova, M.A., Macêdo, J.A.F., Sacramento, E., Pinheiro, A.M.A., Vidal, V.M.P., Breitman, K.K., Furtado, A.L., "Operations over Lightweight Ontologies," in Proc. 11th International Conference on Ontologies, DataBases, and Applications of Semantics - ODBASE 2012b (Sept. 11-12, 2012), Rome. LNCS 7566, 646–663.
- [10] Choksi, A.T., Jinwala, D.C., "A Novel Way to Relate Ontology Classes," in: The Scientific World Journal Volume 2015 (2015), Article ID 724196.
- [11] 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', 82–86, 2010.
- [12] Magalhães, R.C. *Operations over Lightweight Ontologies*. M.Sc. Dissertation, Department of Informatics, PUC-Rio, Rio de Janeiro, Brazil (2015). Available at: <http://www.inf.puc-rio.br/~casanova/Publications/Dissertations-Theses/2015-Romulo.pdf>.
- [13] Noy, N.F., Kunnatur, S., Klein, M. and Musen, M.A., "Tracking changes during ontology evolution," in Sheila A. McIlraith, Dimitris Plexousakis and Frank van Harmelen. In: Proc. 3rd International Semantic Web Conference, 259–273, Hiroshima, Japan, 2004.
- [14] Noy, N. F. and Musen, M. A., "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment," in Seventeenth National Conference on Artificial Intelligence (AAAI-2000), 2000.
- [15] Otero-Cerdeira, L., Rodríguez-Martínez, F. J., Gomez-Rodriguez, A., "Ontology matching: a literature review Expert Systems with Applications" in: Expert Systems with Applications Volume 42, Issue 2, 1 February 2015, Pages 949–971
- [16] Pinheiro, A.M.A., "OntologyManagement Tool – Uma Ferramenta para Gerenciamento de Ontologias como Teorias Lógicas". M.Sc. Dissertation, Dept. Computing, UFC, 2013.
- [17] Prateek, J., Hitzler, P., Sheth, A.P., Verma, K., Yeh, P.Z., "Ontology alignment for linked open data," in: Proc. 9th Int'l. Semantic Web Conf. Springer-Verlag, 402–417, 2010.
- [18] Ramos, J. A. "Mezcla automática de ontologías y catálogos electrónicos," Final YearProject. Facultad de Informática de la Universidad Politécnica de Madrid. Spain, 2001.
- [19] Tury, M. and Bielíková, M., "An Approach to Detection Ontology Changes" in: Proc. 6th Internacional Conference on Web Engineering, 14, New York, NY, USA. ACM Press, 2006.
- [20] Volz, R., Oberle, D. and Studer, R., "Implementing Views for Lightweight Web Ontologies," in: Proc. of Int'l Database Engineering and

Application Symposium - IDEAS, 160-169, Hong Kong, China. IEEE Computer Society, 2003.

- [21] Wang, Z., Zhang, X., Hou, L., Li, J. "RiMOM2: A Flexible Ontology Matching Framework," in: Proc. ACM WebSci'11, Koblenz, Germany, 1-2, 2011.

Referenced Ontologies:

- [22] Parliament Ontology owl description file can be found at: <http://www.inf.puc-rio.br/~rmagalhaes/projetos/ParliamentOntology.owl>
- [23] Parliament Ontology rdf file with data can be found at: <http://www.inf.puc-rio.br/~rmagalhaes/projetos/ParliamentOntology.rdf>
- [24] Relationship Ontology latet rdf file can be found at: <http://vocab.org/relationship/>

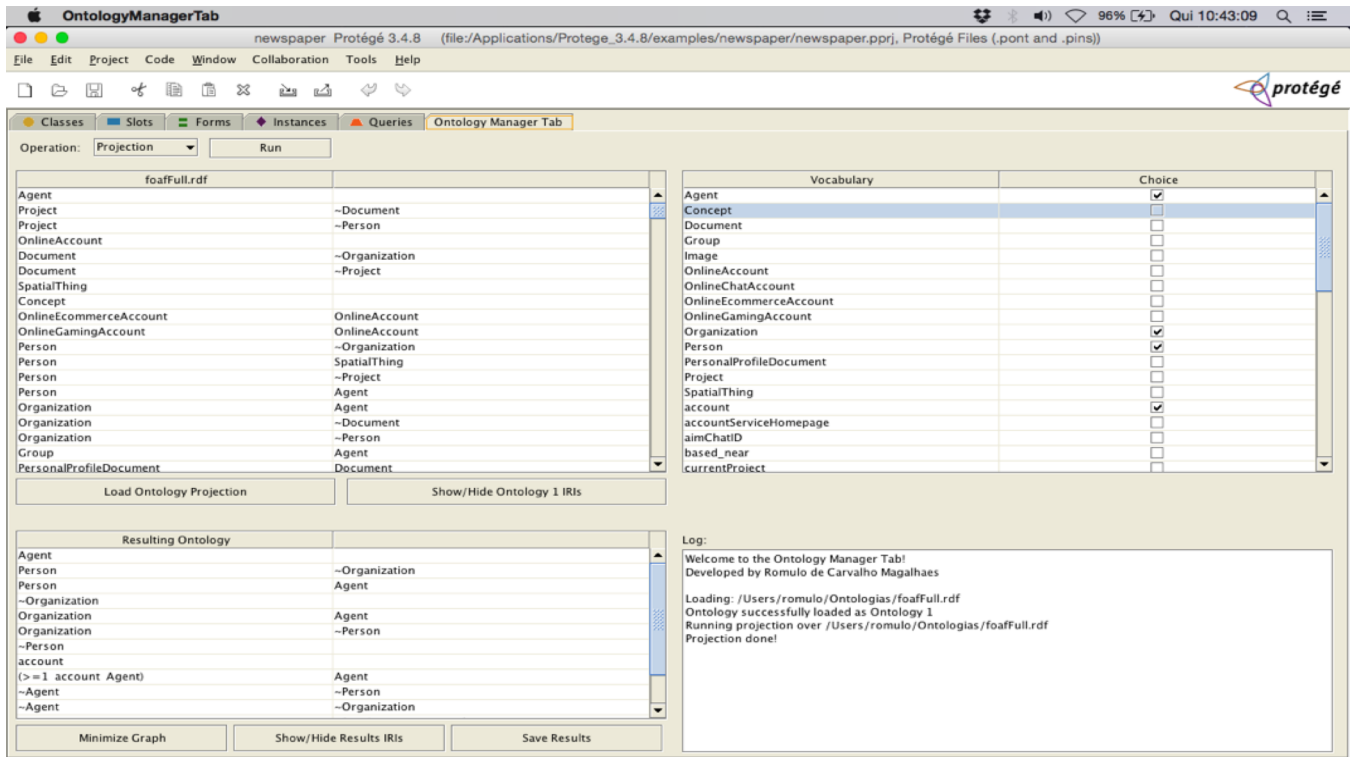


Figure 1 - Projection of some classes of the FOAF ontology.

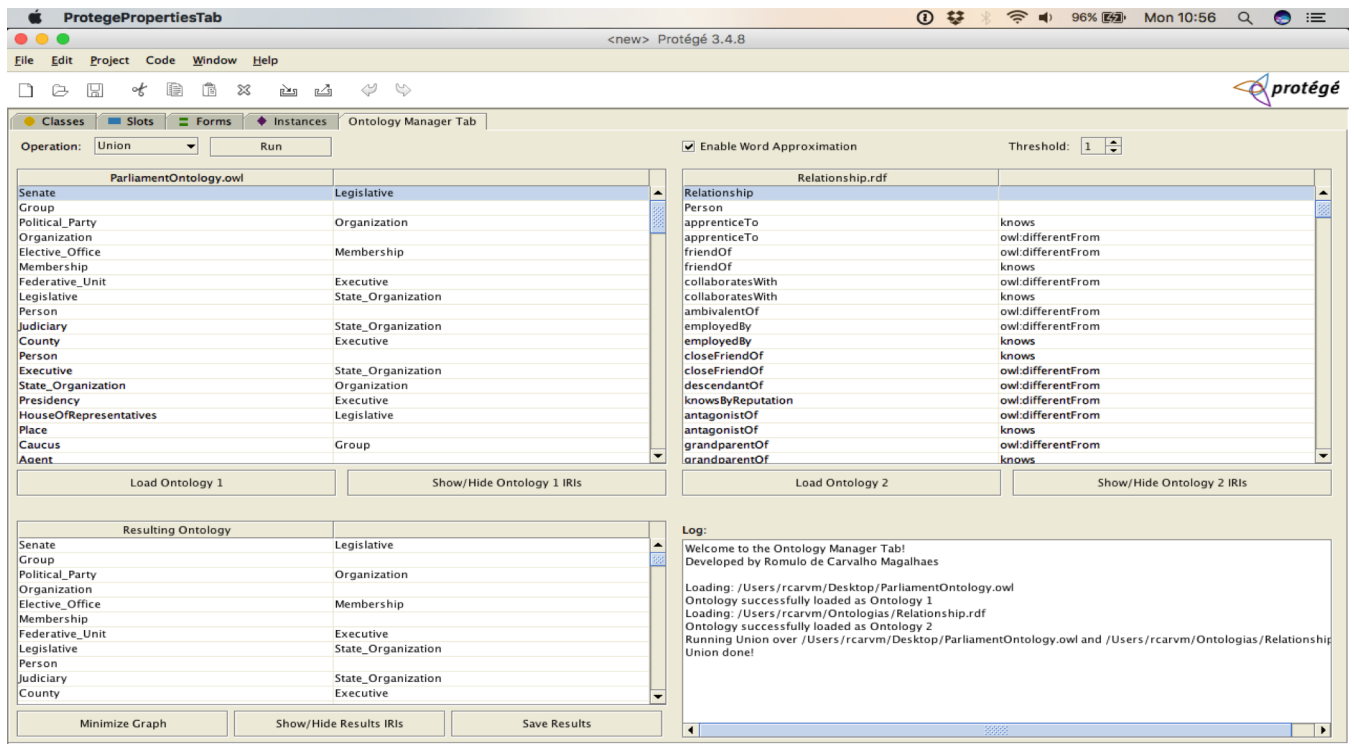


Figure 2 - Union of Parliament and Relationship ontologies.