Pontifícia Universidade Católica
do Rio de Janeiro

Rômulo de Carvalho Magalhães

Operations over Lightweight Ontologies

Dissertação de Mestrado

Dissertation presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática da PUC-Rio as partial fulfillment of the requirements for the degree of Mestre.

Advisor: Prof. Marco Antonio Casanova

Rio de Janeiro
January 2015

Pontifícia Universidade Católica
do Rio de Janeiro

Rômulo de Carvalho Magalhães

Operations over Lightweight Ontologies

Dissertation presented to the Programa de Pós-Graduação em Informática of the Departamento de Informática do Centro Técnico Científico da PUC-Rio, as partial fulfillment of the requirements for the degree of Mestre.

Prof. Marco Antonio Casanova
Advisor
Departamento de Informática – PUC-Rio

Prof. Antonio Luz Furtado
Departamento de Informática – PUC-Rio

Prof. Edward Hermann Haeusler
Departamento de Informática - PUC-Rio

Prof. José Eugenio Leal
Coordinator of the Centro
Técnico Científico – PUC-Rio

Rio de Janeiro, January 30th, 2015

Rômulo de Carvalho Magalhães

Graduated in Computer Engineering from Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro - Brazil in 2009. He joined the Master in Informatics at Pontifical Catholic University of Rio de Janeiro (PUC-Rio) in 2013.

Bibliographic data

de Carvalho Magalhães, Rômulo

Operations over Lightweight Ontologies/ Rômulo de Carvalho Magalhães; advisor: Marco Antonio Casanova. – 2015.

106 f. : il. (color) ; 30 cm

Dissertação (Mestrado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2015.

Inclui bibliografia

1. Informática – Teses. 2. Dados ligados. 3. Ontologias. 4. OWL. 5. RDF. 6. Grafos. 7. Lógica Descritiva (DL) I. Casanova, Marco Antonio. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

I would like to say a special thank you to my parents, Cristina and Walter, for their support and encouragement during all these years of hard study. For raising me and being there when I needed.

To my sister, Amanda, who has always been a great friend even when we both were too busy to be together.

To my advisor, Marco Antonio Casanova, for sharing his knowledge and helping me develop this wonderful work.

To my girlfriend, Patricia, for her support, comprehension, for always listening to me and being always ready to distract me for a while.

To my friend, Daniel, for all his support and for always remembering me to take a break. To all the friends I made during this years of study. Thanks for the support, for sharing your ideas, experiences and always trying to help.

To all the professors and staff from the Computer Science Department. Thanks for all your help and for always being so accommodating.

To PUC-Rio and CAPES for funding my research.

Abstract

Magalhães, Rômulo; Casanova, Marco Antonio (Advisor). **Operations over Lightweight Ontologies.** Rio de Janeiro, 2015. 106p. MSc. Dissertation – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This work addresses ontology design problems by treating ontologies as theories and by defining a set of operations that map ontologies into ontologies, including their constraints. The work first summarizes the base knowledge needed to define the class of ontologies used and proposes four operations to manipulate them. It then shows how the operations work and how they may help design new ontologies. The core of this work is describing the implementation of the operations over a Protégé plug-in, detailing the architecture and including case-use examples.

Keywords

Description Logics; Linked Data; Ontologies; OWL; RDF

Resumo

Magalhães, Rômulo; Casanova, Marco Antonio. **Operações sobre Ontologias Leves.** Rio de Janeiro, 2015. 106p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este trabalho aborda problemas de projeto de ontologias tratando-as como teorias e definindo um conjunto de operações que mapeiam ontologias em ontologias, incluindo suas restrições. Inicialmente, o trabalho resume o conhecimento básico necessário para definir a classe de ontologias utilizada e propõe quatro operações para manipular ontologias. Em seguida, mostra o funcionamento destas operações e como elas podem ajudar na criação de novas ontologias. O cerne do trabalho mostra a implementação destas operações em um plug-in do Protégé, detalhando sua arquitetura e incluindo casos de uso.

Palavras-chave

Lógica de Descrição; Dados Interligados; Ontologias; OWL; RDF

Table of Contents

1 Introduction 12

1.1 Web of Data 12

1.2 Motivation 12

1.3 Goal and Contributions 13

1.4 Dissertation Structure 15

2 Theoretical Basis 16

2.1 Linked Data Technology 16

2.2 A Brief Review of Description Logic 20

2.3 Lightweight Languages 22

2.4 Example 23

2.5 Summary 25

3 Related Work 26

3.1 Overview 26

3.2 Ontology Reuse 26

3.3 Ontology Versioning 28

3.4 Ontology-Based Data Integration 29

3.5 Summary 33

4 Constraint Graphs 34

4.1 Introduction 34

4.2 Constraint Graph Representation 34

4.3 Constraint Graph Basic Functions 37

4.4 Summary 39

5 Implementation of the Operations 40

5.1 Introduction 40

5.2 Definition of Operations over Lightweight Ontologies 40

5.3 Implementation of Projection 42

5.4 Implementation of Union 44
5.5 Implementation of Intersection 47
5.6 Implementation of Difference 49
5.7 Summary 52

6 OntologyManagerTab – an Ontology Manager Plug-in for Protégé 54
6.1 Introduction 54
6.2 Class Architecture 55
6.3 Software Setup 58
6.4 Software Usage 65
6.4.1. Loading Ontologies 65
6.4.2. The Union Procedure 70
6.4.3. The Intersection procedure 75
6.4.4. The Projection procedure 79
6.4.5. The Difference procedure 83
6.4.6. Minimizing Ontologies 85
6.4.7. Saving Resulting Ontologies 92
6.5 Experiments 97
6.6 Summary 99

7 Conclusion 100
7.1 Contributions 100
7.2 Limitations 100
7.3 Future Work 101

8 Bibliographical References 102

List of Figures

Figure 1. Small piece of FOAF ontology. 24
Figure 2. Basic Procedures 39
Figure 3. Packages Interactions 56
Figure 4. Ontology Package interaction with Main Package 58
Figure 5. Launching Protégé 60
Figure 6. Opening Protégé Project 61
Figure 7. Configuring Protégé Widgets 62
Figure 8. Selecting *OntologyManagerTab* 63
Figure 9. *OntologyManagerTab* 64
Figure 10. Loading version of FOAF Ontology 66
Figure 11. Version of FOAF Loaded 67
Figure 12. Hiding FOAF IRIs 68
Figure 13. FOAF normalized file created 69
Figure 14. O_1 loaded as Ontology 1 71
Figure 15. O_2 loaded as Ontology 2 72
Figure 16. Resulting Ontology for the Union of O_1 and O_2 73
Figure 17. *OntologyManagerTab* asking for second ontology 74
Figure 18. $O_{PhoneCompany1}$ loaded as Ontology 1 76
Figure 19. $O_{PhoneCompany2}$ loaded as Ontology 2 77
Figure 20. Resulting Ontology for the Intersection of $O_{PhoneCompany1}$ and $O_{PhoneCompany2}$ 78
Figure 21. FOAF Ontology loaded as Ontology 1 80
Figure 22. Selection of V_{FF} for the Projection procedure 81
Figure 23. Resulting Ontology for the Projection of V_{FF} over the FOAF ontology 82
Figure 24. Resulting Ontology for the Difference of $O_{PhoneCompany1}$ and $O_{PhoneCompany2}$ 84
Figure 25. O_1 loaded as Ontology 1 and O_2 loaded as Ontology 2 86

Figure 26. result of said Intersection between O_1 and O_2 87
Figure 27. Minimize Graph function over Intersection between O_1 and O_2 88
Figure 28. Union between O_3 and O_4 90
Figure 29. Graph Minimization for the Union between O_3 and O_4 91
Figure 30. Saving the Resulting Ontology for the Projection of V_{FF} over the FOAF ontology 93
Figure 31. Ontology saved as “FoafFacebookOntologyNormalized.owl” 94
Figure 32. Loading saved ontology 95
Figure 33. Recently saved ontology loaded 96
Figure 34. Generated OWL file, “FoafFacebookOntologyNormalized.owl”, in a text editor 97

List of Tables

Table 1. Common constraints used and their tautologies 23
Table 2. Constraints for FOAF ontology on Figure 1. 24
Table 3. Comparative analysis between the applications for ontology integration. 33
Table 4. Constraints of Ontology FF 44
Table 5. Constraints Σ_U of Ontology O_U. 45
Table 6. Constraints for $O_{PhoneCompany1}$, $O_{PhoneCompany2}$ and their Union $O_{PhoneCompany3}$ 46
Table 7. Constraints Σ_{int} of Ontology O_{int} 48
Table 8. Constraints for $O_{PhoneCompany1}$, $O_{PhoneCompany2}$ and their Intersection $O_{PhoneCompany3}$ 49
Table 9. Constraints Σ_D of Ontology O_D 51
Table 10. Constraints for $O_{PhoneCompany1}$, $O_{PhoneCompany2}$ and their Difference $O_{PhoneCompany3}$ 52
Table 11. Experiments Processing Times in seconds 98

1
Introduction

1.1 Web of Data

Currently, the World Wide Web contains an almost immeasurable amount of information, which grows daily as its popularity increases throughout the world, thus becoming increasingly composed of heterogeneous data sources, bearing in mind that there is no standard for their publication. It is in this context that arises the concept of *Web of Data*, which defines an approach to publish, retrieve and describe distributed data on the Web.

The *Web of Data* is based on the principles of Linked Data proposed by (Tim Berners-Lee, 2006), which defines a group of best practices to publish and connect structured data on the Web. For this purpose, the Linked Data principles use Semantic Web technologies such as URI (Unified Resource Identification), RDF triples (Resource Description Framework) (Klyne, G. Et al., 2010) and OWL (Web Ontology Language) (McGuinness, DL, Harmelen, F ., 2010), which will be explained in depth in Chapter 2.

From a broader perspective, the Linked Data principles suggest a way to publish databases on the Web that facilitates interoperability. In fact, the problem of interoperability between databases has persisted since the 80's, if not earlier, without appropriate approaches (Haas and Carey, 2003). Furthermore, 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 proned task, which lies at the core of the interoperability issue.

1.2 Motivation

The purpose of this dissertation is to develop a tool to assist the *domain specialist* in the process of managing ontologies for data publication.

Many tools have been developed with the aim of assisting the process of developing an ontology based on others, by heuristics or techniques for ontology alignment. However, such tools do not properly assist the user in developing a new ontology that represents a correct understanding of the semantics of the involved data sources, expressed by the logical constraints of the original ontologies. As a matter of fact, this requires propagating the original constraints to the new ontology.

In the Web of Data, the main obstacle to the integration and interoperability of existing data sources emerges from the fact that these sources are described by ontologies developed independently. This happens most often because of a lack of concern while reusing terminologies of the most popular and widely disseminated ontologies on the Web, which in turn hinders the recognition of connections between distinct data sources.

Furthermore, we must consider semantic conflicts that occur when the same term represents different concepts (Noy, NS, 2004), in other words, when the same symbol accounts for two distinct concepts. As an example, we may consider the term *foot* in two distinct data sources, one of the healthcare domain and the other of the measurement units domain, representing completely different concepts. In the first source, "foot" represents a human body part, whereas in the second "foot" is a measurement unit, mostly used in countries that were British colonies.

1.3 Goal and Contributions

In order to address the problems outlined in Section 1.2, this work will consider ontologies as logical theories, composed of vocabularies and constraints, and will define algebraic operations (**Projection, Union, Intersection** and **Difference**) over one or two ontologies, as proposed in (Casanova et al., 2012), using the concept of Constraint Graphs in order to enable the integration and interoperability of the data sources that follow the Linked Data principles. These algebraic operations permit defining new ontologies out of existing ones and take into account the semantics of the involved ontologies.

In more detail, we define an *ontology* as a pair $O=(V,\Sigma)$ such that V is a *vocabulary* and Σ is a set of *constraints* in V. The *theory of* Σ is the set of all constraints that are logical consequences of Σ. We emphasize that the constraints in Σ capture the semantics of the terms in V and must, therefore, be brought to the foreground. The theory of Σ identifies the constraints that are implicitly defined, but which must be considered when using the ontology.

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.

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 Σ_1, but not in the theory of Σ_2, after the terms are appropriately mapped. The *difference* operation answers this question.

Another variant of ontology comparison is the problem of analyzing what changed from one version of an ontology to the other. Difference is especially useful here.

Also, if the designer wishes to use only part of a given ontology, he must define a set W containing just a few terms from the vocabulary V and detect which constraints in the theory of Σ references only to said terms. For that purpose we have the *Projection* operation.

Finally, if the designer wants to combine $O_1=(V_1,\Sigma_1)$ and $O_2=(V_2,\Sigma_2)$, he should again create a mapping between their vocabularies and detect which constraints hold in both theories and afterwards create a new ontology with all terms and constraints of the originals, taking into account the mapping not to add duplicates. To that end we have the *Union* operation.

The main contribution of this work is the development of a software tool as a plug-in for Protégé, the most popular platform for editing ontologies, which will implement the algebraic operations to assist the domain specialist in managing ontologies. However, this software tool will be completely independent from any of the Protégé libraries, using it as a graphical user interface (GUI) Front-End

only. Thus, it will be possible to adapt the tool to any other ontology editor in the future, provided that the necessary integration procedures are correctly performed.

1.4 Dissertation Structure

This dissertation is structured as follows. Chapter 2 presents the basic concepts related to this work: Linked Data Technology, Description Logic and the conceptual schemas adopted. Chapter 3 describes related work, qualifying and comparing the proposed tool with others. Chapter 4 presents the concept of constraint graph and the algorithms developed to build it. Chapter 5 addresses the implementation adopted for each operation (**Projection, Union, Intersection** and **Difference**). Chapter 6 details the **OntologyManagerTab** with its class architecture, a Setup Guide and use examples for its functionalities. Finally, Chapter 7 discusses the limitations of this work and suggests possible future work.

2
Theoretical Basis

This chapter provides an overview of the main concepts related to this dissertation. Section 2.1 introduces the key concepts of the Linked Data Technology. Section 2.2 presents a brief review of Description Logic. Section 2.3 describes the family of Description Logic languages adopted in this work. Finally, Section 2.4 presents an example.

2.1 Linked Data Technology

Linked Data is a set of better practices for consumption and publication of structured data in the Web, with the goal to establish connections between items of different data sets to form a single global space of data (Heath, T. and Bizer, C., 2011). These better practices were initially proposed by (Berners-Lee, 2007) and became known as the principles of Linked Data. They are:

1. The usage of URIs as names for objects.

2. The usage HTTP URIs in such a manner that applications and users can follow them.

3. Provide useful information through standards (RDF, SPARQL), when a URI is followed.

4. Include RDF declarations that Link URIs between themselves, allowing the extraction of new relationships.

These are the principles that supply the basis for publishing and interconnecting structured data in the Web. The open standards adopted in Linked Data are widely known and will be detailed below.

URI – Uniform Resource Identifier

A *Uniform Resource Identifier* (URI) is a sequence of characters that identifies a physical or abstract resource (Berners-Lee T. et al., 2005). Since the most com-

mon form of URI is the *Uniform Resource Locator* (URL), also known as a Web address, URLs can be used to identify things in the Web.

To differentiate between URIs that represent Web pages and those that represent things in the Web, two distinct forms of URI were defined: *hash URI* (Berners-Lee, T., 1994) and *slash URI* (Berners-Lee, T., 2007). Usually, hash URIs are used to identify things and are composed by: (document) # (term to be introduced in the document), as an example we have

http://www.w3.org/2002/07/owl#Thing

On the other hand, the slash URI defines an URL, as in the following example:

http://purl.org/dc/elements/1.1/title

Ontology

A widely used definition of ontology in the Web Semantics literature is (Gruber et al, 1993): "An ontology is defined as a formal, explicit specification of a shared conceptualization".

According to (Breitman et al, 2006), in Gruber's definition, the term "conceptualization" expresses an abstract model; "explicit" means that the elements must be clearly defined; and "formal" indicates that the specification must be processable by a machine. Therefore, it is possible to conclude that, in Gruber's definition, an ontology is a representation of a knowledge domain, where a set of objects and their relationships are described by vocabularies.

Currently, ontologies can be textually represented in XML based languages, such as RDF, RDF Schema (RDF-S) and OWL.

RDF – Resource Description Framework

The Resource Description Framework (RDF) describes a "standard model" for data exchange in the Web. RDF is used to represent metadata resources in the Web. Most ontology definition languages are based on RDF.

RDF is based on the idea that resources are identified using URIs and are described in terms of simple properties and property values, thereby creating sets of triples, composed of a subject, a predicate and an object, where:

1. The subject denotes a resource (identified by a URI);
2. The predicate names a property of the resource;

3. The object indicates the property value, which can be a literal (represented by an integer or a string, for example), or even a URI identifying another resource.

As an example, we have the following declaration:

http://www.inf.puc-rio.br/ hasProfessor Marco A. Casanova

where

- Subject: URL http://www.inf.puc-rio.br/

- Predicate: “hasProfessor”

- Object: “Marco A. Casanova”

RDF Schemas extend RDF to include constructors for classes, subclasses, properties and sub-properties.

RDF Vocabulary

An *RDF vocabulary* supplies domain-specific terms to describe classes of resources and the types of relationship between them. Depending on the expressive power, vocabularies can be classified from *taxonomies* to *ontologies* (McGuinness, D.L., 2002).

To increase the interoperability between applications, it is recommended to reuse terms of RDF vocabularies, which are widely used. As examples of vocabularies well diffused throughout the community, that should be used whenever possible, we can name:

- **Dublin Core Metadata Initiative** (DCMI)[1] seeks to describe attributes of general metadata, such as: creator, data, subject and description, among others.
- **Friend-of-Friend** (FOAF)[2] defines terms to describe people, their activities and their relationships with other people, objects and websites (Brickley, D., Miller, L., 2010).
- **Description of a Project** (DOAP)[3] defines terms to describe software projects, particularly Open Source ones.

[1] https://github.com/edumbill/doap/
[2] http://www.foaf-project.org/
[3] http://semanticweb.org/wiki/DOAP

- **Music Ontology**[4] defines terms to describe various concepts related to music, such as: artists, albums, tracks and others.

OWL - Web Ontology Language

The *Web Ontology Language* (OWL), standardized by W3C, is a computational, logic-based language such that knowledge expressed in it can be used to verify the consistency of the knowledge or to make implicit knowledge, explicit.

OWL offers specific constructors for basic ontological concepts, such as classes, instances, properties and cardinality restrictions, as well as constructors for the formalization of more complex relationships, such as equivalency, union and intersection. Originally, OWL had three sublanguages of increasing expressiveness:

1. OWL Lite – it supports the creation of classification hierarchies and some simple constraints. It supports cardinality restrictions with values of 0 or 1. The objective of this sublanguage is to offer support to the migration from taxonomies to the format of ontologies.
2. OWL DL (Description Logic) – it supports all the constructors offered for OWL language; it offers the maximum expressiveness within the bounds of the computational completeness and decidability of Description Logic. This sublanguage possesses constructors, which are more complex than those of OWL Lite, enabling the modeling of classes by means of the operators of union, intersection and complement, besides representing class disjunctions.
3. OWL Full – allows the maximum expressiveness and the syntactic freedom of RDF, but with no computational guarantees.

These three sublanguages have distinct target audiences and the choice of one of them is crucial to the success of the Linked Data application.

[4] http://musicontology.com/

2.2 A Brief Review of Description Logic

A (Description Logic) *language* $\mathcal{L}$ is characterized by an *alphabet* $\mathcal{A}$, consisting of: a set of *atomic concepts*; a set of *atomic roles*; the *universal concept*, also known as *top*, denoted by $\top$; the *empty concept*, also know as *bottom*, denoted by $\bot$; and the *universal role* also denoted by $\top$; and the *empty role*, also denoted by $\bot$.

The set of *role descriptions* of $\mathcal{L}$ is inductively defined as follows:

- An atomic role and the universal and empty roles are role descriptions.
- If p is a role description, then the following expressions are role descriptions:

  p^{-} (the inverse of p)

  $\neg p$ (the negation of p)

The set of *concept descriptions* of $\mathcal{L}$ is inductively defined as follows:

- The atomic concept, the universal and the empty concept are concept descriptions.
- If *e* is a concept description, *p* is a role description and *n* is a positive integer, then the following expressions are concept descriptions:

  $\neg e$ (negation)

  $\exists p$ (restricted existential quantification)

  $(\leq n\ p)$ (Maximum cardinality restriction)

  $(\geq n\ p)$ (Minimum cardinality restriction)

An interpretation $\boldsymbol{s}$ for the symbols of the alphabet $\mathcal{A}$ consists of a non-empty set $\Delta^{\boldsymbol{s}}$, the *domain* of $\boldsymbol{s}$, whose elements are called *individuals*, and an interpretation function, also denoted $\boldsymbol{s}$, where:

$\boldsymbol{s}(\top) = \Delta^{\boldsymbol{s}}$, if $\top$ denotes the universal concept

$\boldsymbol{s}(\top) = \Delta^{\boldsymbol{s}} \times \Delta^{\boldsymbol{s}}$, if $\top$ denotes the universal role

$\boldsymbol{s}(\bot) = \varnothing$, if $\bot$ denotes the empty concept or the empty role

$\boldsymbol{s}(A) \subseteq \Delta^{\boldsymbol{s}}$, for each atomic concept A in $\mathcal{A}$

$\boldsymbol{s}(P) \subseteq \Delta^{\boldsymbol{s}} \times \Delta^{\boldsymbol{s}}$ for each atomic role P in $\mathcal{A}$

The function $\boldsymbol{s}$ is extended for concept and role descriptions of $\mathcal{L}$ as follows (where *e* is a concept description and *p* is a role description):

$\boldsymbol{s}(p^{-}) = \boldsymbol{s}(p)^{-}$ (the inverse of $\boldsymbol{s}(p)$)

$\boldsymbol{s}(\neg p) = \Delta^{\boldsymbol{s}} \times \Delta^{\boldsymbol{s}} - \boldsymbol{s}(p)$ (the complement of s(p) in relation to $\Delta^{s} \times \Delta^{s}$)

$\mathbf{s}(\neg e) = \Delta^{s} - \boldsymbol{s}(e)$ (the complement of s(e) in relation to Δ^{s})

$\boldsymbol{s}(\exists p) = \{I \in \Delta^{\boldsymbol{s}} / (\exists J \in \Delta^{s})((I,J) \in \boldsymbol{s}(p)\}$

(the set of individuals I such as $\boldsymbol{s}(p)$ maps I to some individual J)

$\boldsymbol{s}(\geq n\ p) = \{I \in \Delta^{s} / |\{J \in \Delta^{\mathbf{s}} / (I,J) \in \boldsymbol{s}(p)\}| \geq n\}$

(the set of individuals I such as $\boldsymbol{s}(p)$ maps I to at least n distinct individuals)

$\boldsymbol{s}(\leq n\ p) = \{ I \in \Delta^{\mathbf{s}} / |\{J \in \Delta^{s} / (I,J) \in \boldsymbol{s}(p)\}| \leq n\}$

(the set of individuals I such as $\boldsymbol{s}(p)$ maps I to at most n distinct individuals)

A *formula* of $\mathcal{L}$ is an expression of the form $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$, called an *inclusion*. Other subtypes of formulas are: $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ is a *concept inclusion* iff $\boldsymbol{u}$ and $\boldsymbol{v}$ are both concept descriptions, and $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ is a *role inclusion* iff $\boldsymbol{u}$ and $\boldsymbol{v}$ are both role descriptions. We also define a formula of the form $\boldsymbol{u} \mid \boldsymbol{v}$, called a *disjunction*, or of the form $\boldsymbol{u} \equiv \boldsymbol{v}$, called an *equivalence*, where $\boldsymbol{u}$ and $\boldsymbol{v}$ are either concept descriptions or role descriptions of $\mathcal{L}$. A disjunction formula $\boldsymbol{u} \mid \boldsymbol{v}$ is logically equivalent to the inclusion $\boldsymbol{u} \sqsubseteq \neg \boldsymbol{v}$ and an equivalence $\boldsymbol{u} \equiv \boldsymbol{v}$ is logically equivalent to $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ and $\boldsymbol{v} \sqsubseteq \boldsymbol{u}$.

An interpretation of $\boldsymbol{s}$ for $\mathcal{L}$ satisfies $\boldsymbol{u} \sqsubseteq \boldsymbol{v}$ iff $\boldsymbol{s}(\boldsymbol{u}) \subseteq \boldsymbol{s}(\boldsymbol{v})$; $\boldsymbol{s}$ satisfies $\mathbf{u} \mid \mathbf{v}$ iff $\boldsymbol{s}(\boldsymbol{u}) \cap \boldsymbol{s}(\boldsymbol{v}) = \varnothing$, in other words, if $\boldsymbol{s}(\boldsymbol{u})$ and $\boldsymbol{s}(\boldsymbol{v})$ are disjoint sets; and $\boldsymbol{s}$ satisfies $\boldsymbol{u} \equiv \boldsymbol{v}$ iff $\boldsymbol{s}(\boldsymbol{u}) = \boldsymbol{s}(\boldsymbol{v})$.

Let σ and σ' be two inclusions of $\mathcal{L}$ and Σ be a set of inclusions of $\mathcal{L}$. Assume that σ is of the form $u \sqsubseteq v$. We say that:

- $\boldsymbol{s}$ *satisfies* σ or $\boldsymbol{s}$ is a *model* of σ, denoted $\boldsymbol{s} \vDash \sigma$, iff $\boldsymbol{s}(u) \subseteq \boldsymbol{s}(v)$.
- $\boldsymbol{s}$ *satisfies* Σ or $\boldsymbol{s}$ is a *model* of Σ, denoted $\boldsymbol{s} \vDash \Sigma$, iff $\boldsymbol{s}$ satisfies all inclusions in Σ.
- σ is *valid*, denoted $\vDash \sigma$, iff any interpretation for V satisfies σ.
- σ and σ' are *tautologically equivalent* iff any model of σ is a model of σ' and vice-versa.
- Σ *logically implies* σ, or σ is a *logical consequence* of Σ, denoted $\Sigma \vDash \sigma$, iff any model of Σ satisfies σ.
- Σ is *satisfiable* or *consistent* iff there is a model of Σ.

The *theory* of Σ in V, denoted $\tau[\Sigma]$, is the set of all inclusions in V that are logical consequences of Σ. We say that two sets of inclusions, Γ and Θ, are *equivalent*, denoted $\Gamma \equiv \Theta$, iff $\tau[\Gamma]=\tau[\Theta]$.

Finally, an *ontology* is a pair $\boldsymbol{O}=(V,\Sigma)$ such that V is a finite alphabet, called the *vocabulary* of $\boldsymbol{O}$, whose atomic concepts and atomic roles are called *classes* and *properties* of $\boldsymbol{O}$, respectively, and Σ is a set of inclusions in V, called the *constraints* of $\boldsymbol{O}$. Two ontologies $O_1 = (V_1,\Sigma_1)$ and $O_2 = (V_2,\Sigma_2)$ are *equivalent*, denoted $O_1 \equiv O_2$, iff Σ_1 and Σ_2 are equivalent.

2.3 Lightweight Languages

In this dissertation, the family of lightweight languages was adopted, which is equivalent to the family of DL-Lite core with number restrictions (Artale, A. et al, 2009). Lightweight languages are sufficiently expressive to capture the constructors of the main data modeling languages, such as UML and ER (Borgida and Brachman, 2003). They support classes, datatype properties, object properties, minimum and maximum cardinality (minCardinalities and maxCardinalities), inverse functional property (InverseFunctionalProperties) that captures simple keys, class subset and class disjointness. Furthermore, lightweight languages allow the use of decision procedures that exploit the structure of constraint sets (Casanova et al., 2011).

A *lightweight language* is defined in Section 2.2, except that the formula types are listed in Table 1, called *lightweight inclusions*. All inclusions that appear in the "Abbreviated form" column can be rewritten equivalently as shown in the "Unabbreviated form" column, which indicates the *normalized form* of the inclusion. As observed in (Casanova et al., 2011), the normalized form avoids the use of the existential quantifier and maximum cardinality constraints. Furthermore, negated descriptions appear only on the right side of the normal form.

Finally, a *lightweight ontology* is an ontology which constraints are lightweight inclusions. From this point on, we will use interchangeably the terms 'lightweight inclusion' and 'lightweight 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 $C \sqsubseteq (\geq k\ P^{-})$	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		$C \sqsubseteq D$	Each individual in C is also in D, that is, class C denotes a subset of class D
Disjointness Constraint	$C \mid D$	$C \sqsubseteq \neg D$ and $D \sqsubseteq \neg C$	No individual is in both C and D, that is, classes C and D are disjoint

Table 1. Common constraints used and their tautologies

2.4 Example

In this section, we exemplify the concepts defined so far, using the fragment of the FOAF ontology shown in Figure 1. The Friend-of-a-Friend (FOAF) ontology defines terms to describe people, their activities and their relationships with other people, objects and websites. Figure 2 has the normalized constraints for this fragment of FOAF, where the first column contains the image and domain constraints, the second column, the minimum and maximum cardinality restrictions and the third, the subset and disjunction constraints.

For a better understanding of our representation, we now explain each constrain in Table 2. From the first column we have:

- $\exists name \sqsubseteq Person$ – *name* is an atomic role which is a property of *Person* (the domain of *name*).
- $\exists name^{-} \sqsubseteq String$ – $name^{-}$ has *String* as its domain, thus the property *name* has *String* as its range.

From the second column, we have the minimum and maximum cardinalities for the property *name*:

- $Person \sqsubseteq (\leq 1\ name)$ – maximum cardinality restriction for the property *name* in *Person* with value 1.

- *Person* ⊑ (≥1 *name*) – minimum cardinality restriction for the property *name* in *Person* with value 1.

From the third column, we have the subset and disjointness constraints:

- *Person* ⊑ ¬*Organization* – indicates that the classes *Person* and *Organization* are disjoint.
- *Organization* ⊑ ¬*Person* – equivalently indicates that the classes *Person* and *Organization* are disjoint.
- *Group* ⊑ *Agent* – the class *Group* is a subset of the class *Agent.*
- *Person* ⊑ *Agent* – the class *Person* is a subset of the class *Agent.*
- *Organization* ⊑ *Agent* – the class *Organization* is a subset of the class *Agent*.

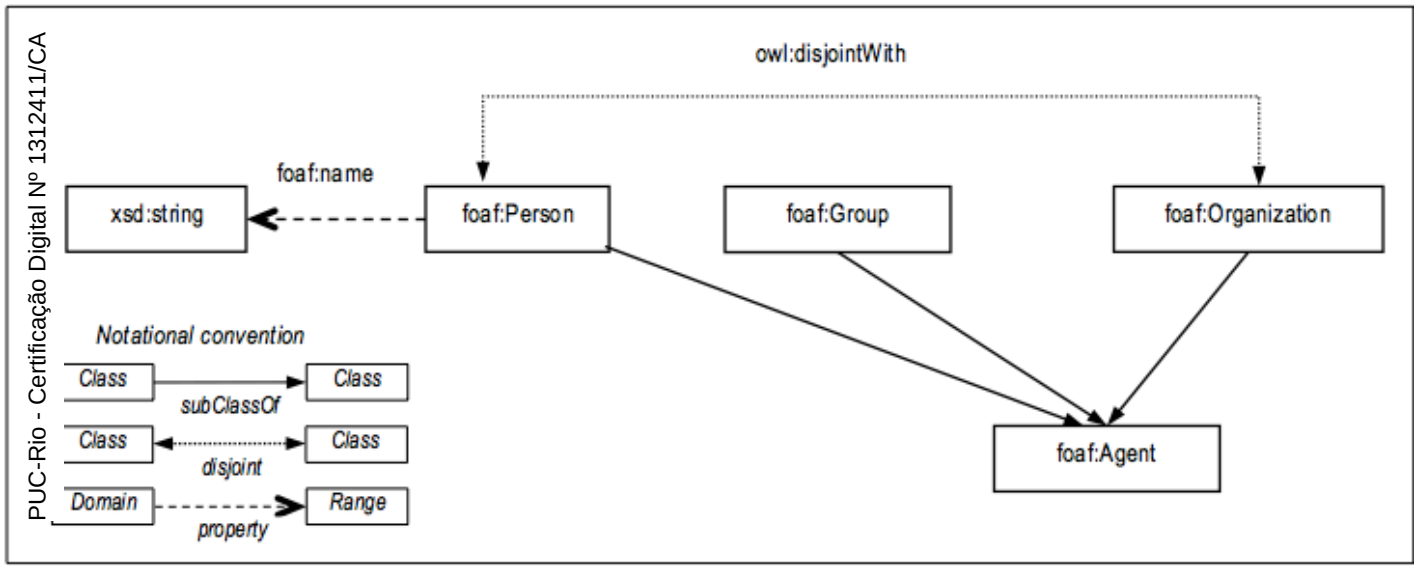

Figure 1. Small piece of FOAF ontology.

Domain and Range Constraints	Cardinality Constraints	Subset and Disjunction Constraints
∃*name* ⊑ *Person* ∃*name*⁻ ⊑ *String*	*Person* ⊑(≤1 *name*) *Person* ⊑(≥1 *name*)	*Person* ⊑ ¬*Organization* *Organization* ⊑ ¬*Person* (equivalent to the previous constraint) *Group* ⊑ *Agent* *Person* ⊑ *Agent* *Organization* ⊑ *Agent*

Table 2. Constraints for FOAF ontology on Figure 1.

2.5 Summary

This chapter presented the main concepts related to this dissertation. Section 2.1 introduced the key concepts of the Linked Data Technology. Section 2.2 presented a brief review of Description Logic. Section 2.3 described the family of Description Logic languages adopted in this work. Finally, Section 2.4 presented a simple example.

3
Related Work

3.1 Overview

Ontology management is an essential aspect for the development of applications using Linked Data technologies. It can be defined as a set of methods and techniques required to effectively handle multiple ontologies derived from heterogeneous and defined data sources to comply with the most diverse purposes (Peter Haase *et al*, 2003).

There are several studies about this particular topic. In this chapter, we will separate them in areas of focus and address their relationship with the current work. Furthermore, we will discuss other works related to the areas of ontology reuse, ontology versioning and ontology-based data integration. Each area corresponds to a section. The final section contains considerations about the work presented in this dissertation.

3.2 Ontology Reuse

According to Bizer *et al.* (2009), ontology reuse can be defined as the process in which a reference ontology, namely a widely used, consolidated and tested ontology is used as an input to generate a new one. This process can take two approaches, the *syntactic reuse* and the *semantic reuse*. The syntactic reuse consists of importing a set of terms of an existing ontology vocabulary to the new ontology. The semantic reuse consists of applying the constraints of an existing ontology to the terms of a new one.

The syntactic approach is easily handled by various ontology management programs, including the Protégé (Gennari J. H. *et al*., 2003) for which we built our plug-in. Such programs simply import the needed OWL ontologies by using the declaration tag <owl:import> from the OWL Language.

This approach is quite simple but has its disadvantages. By using <owl:import>, the user is importing the whole ontology, even if he wishes to use only a small part of it, which can cause inconsistencies between the terms of the imported ontology and the current one.

When an ontology imports another one, it effectively dictates that all semantic conditions of the imported ontology are held in the importing ontology. As a result, the *imports* relationship is transitive: if an ontology *A* imports an ontology *B* and *B* imports an ontology *C*, then *A* also imports *C*. Thus, we must be careful with this import closure as not to generate any inconsistencies.

Frequently, the size and complexity of available ontologies exceed the needs of an application, which may need only part of the domain described by the ontology. By using complete ontologies, an application ends up creating computational penalties such as issues in performance, space and processing time, primarily in the use of inference mechanisms.

Hence, if the user wishes to import only a small subset of an ontology, he must perform the costly procedure of carefully identifying which set of axioms represents the knowledge base he wants to import. This is not always a trivial matter, since we have to consider the size of an ontology as well as its number of concepts and constraints, and we must also take into account the relevance of these concepts and constraints to the knowledge base the user wants to import.

Ontology reuse is highly recommended in the development of new ontologies and is part of the best practices in Web Semantics. It reduces duplication efforts, the cost and complexity of creating a new ontology from the ground up; Furthermore, since ontologies are understood as means for shared knowledge conceptualization, reusing existing ontological sources increases application interoperability both at the syntactic and at the semantic level.

There are several tools that manage ontology reuse through the extraction of a fragment of an ontology to create a new one. In (Volz R. *et al*., 2003), this issue is treated similarly to the definition of a database view, described by a query over the underlying database. However, other approaches consider that the extraction of a fragment of an ontology is conceptually different from the definition of a database view, since an ontology may contain other types of elements, such as classes, relationships and constraints, that cannot be defined in a query.

Considering this last case, Noy and Musen (2004) presented the concept of *traversal view*, which is a view defined by the user where he specifies the central concept or the relevant concepts, the relationships employed to find concepts to be included, as well as the depth of the traversal. Based on this approach, they developed a Protégé plug-in.

In our work, we adopted the approach of supporting ontology reuse by extracting fragments of an ontology using the **projection** operation based on a selection of vocabulary terms from the original ontology.

3.3 Ontology Versioning

Ontology versioning relates to the fact that there might be multiple variants of an ontology, since the ontologies may be developed in a collaborative manner by several users that improve and perfect them according to the latest needs and uses, thus forming a derivation tree.

In this scenario, the applications and users need to handle the different versions of a particular ontology over time. Klein et al. (2002) developed an ontology versioning control tool that supports ontology updates and their effects through the creation and maintenance of its various versions.

Usually, older and newer versions of the same ontology are provided by developers, but with no efficient way to highlight the differences between the versions. To detect the differences, in most cases, one compares two versions of the same ontology, identifying existing differences between them. Simple differences are those that do not affect the ontology structure, such as changing class and property names. Complex differences include updating the class hierarchy or the semantic concepts (constraint modification).

There are several tools and mechanisms that address the detection of ontology updating in an automatic or semi-automatic manner (with or without user intervention). Among these, we have PROMPTDIFF (Noy N.F. *et al*, 2004), which identifies structural changes in the ontology and allows users to accept or reject each alteration. This helps versioning control in an ontology collaborative development process.

We also have the OntoDiff tool (Tury, M. e Bieliková. M., 2006), which automatically detects modifications between the structure and the contents of two versions of an ontology. This tool uses relative text comparison to identify terms that were added, removed and modified.

The approach presented in our work to handle ontology versioning is based on the **difference** operation. This operation detects which constraints are held in the first ontology, but not in the second.

3.4 Ontology-Based Data Integration

Ontology-based data integration involves the use of ontologies to effectively combine data or information from multiple heterogeneous sources.

Data integration systems provide integrated access to heterogeneous and distributed sources of data (Langegger, A.A., 2010). The primary advantage of these systems is to enable the user to obtain a complete and consistent view of all existing data without the need to access each source separately. According to (Lenzerini, 2002), the data integration frameworks usually follow two classical approaches, *materialized* or *virtual*.

In the materialized approach, the data is retrieved from multiple sources and imported into local data repositories also known as data warehouses. In this approach, the queries are performed over a materialized database thus performing better in relation to the virtual approach. However, considering that the data repositories are dynamic and autonomous, it is extremely costly to keep a data warehouse up to date. Thus, several algorithms are required in order to constantly update and extract relevant information from distributed databases. The primary disadvantage of this approach is the need to keep the data warehouse always up to date, which is costly in terms of processing.

In the virtual approach, the data is retrieved directly from the source when the integration system needs to answer a query. In other words, the integration systems send queries directly to the data sources and then the individual results obtained are integrated to compose the answer to the submitted query. The main advantage of this approach is to ensure that the accessed data is always up to date, however, it is known that the costs of query processing and constant access to the

data sources are very high and must be considered as critical to any applications using said approach.

Therefore, it is extremely important that the data integration application uses the most suitable approach to its means, considering its architecture and features. Most of the current data integration systems adopt a virtual approach with the objective of providing integrated sharing of information presented across multiple data sources. In the virtual approach, each independent data source is represented by its own ontology, thus we have several ontologies being integrated to form a global one, which is obtained by the mapping between these various ontologies.

This approach to ontology integration, also known as *union of ontologies*, identifies identical entities among the ontologies that describe the data sources. Then, it builds a new consistent and minimal ontology that corresponds to the representation of the union of all the information from the original data sources.

Considering this integration, schema matching techniques become necessary in order to identify the singular entities amongst the ontologies. Several studies and applications have been developed over these techniques (Rahm and Bernstein, 2001). According to (Shvaiko, P. and Euzenat., J., 2004), there are two main classifications for schema matching techniques: those based on the schema elements and those based on the structure of the schemas.

Matching techniques based on schema elements perform the alignment of the elements by separately analyzing entities, ignoring their relationships with other entities. In this category, the following techniques are introduced:

- **String based techniques -** use the similarity of the names and descriptions of schema elements; the more similar the strings, the greater the possibility that they represent the same entity.
- **Language based techniques -** consider strings as words in some natural language and exploit the morphological properties of the terms, as the identification of basic word forms and the deletion of articles, prepositions and conjunctions.
- **Constraints based techniques -** analyze the constraints applied to the definitions of the entities, such as their data types, cardinality of the attributes and their declared instances. These techniques are applicable

even if the entities have the same declared instances and the same cardinality for some attributes.

- **Linguistic resources based techniques -** use common knowledge or domain specific thesauri to analyze linguistic relations in the word matching process.

Matching techniques based on the schema structure perform the alignment of elements by analyzing the structure of the entities. In this category, the following techniques are introduced:

- **Graph based techniques –** treat the schemas as a graph structure and identify similar structures between the schemas by analyzing their equivalent parts. Graph matching is a combinatorial problem that can be computationally very expensive and is usually solved by approximated methods (Rahm e Bernstein, 2001).
- **Taxonomy based techniques -** are graph algorithms that consider only specialization, which is used to match concepts that are already similar to try to establish new matches using their neighbors.
- **Logic models based techniques -** are algorithms that consider the semantic interpretation of the model, using deductive methods, such as propositional satisfiability and Description Logic. This approach proposes the decomposition of the graph-matching problem in node matching problems. It uses propositional satisfiability to translate the matching nodes and their possible relations into propositional formulas, with logical operators such as $\sqsubseteq$ and $\equiv$. However, the propositional language used in deduction techniques based on propositional satisfiability is limited in its expressiveness, because it treats only unary predicates. Using description logic, it is possible to treat the binary relationships, such as properties and roles, as well as equivalence ($\equiv$) and subsumption ($\sqsubseteq$).

There are several applications for schema integration developed using these techniques. Most of them are integrated with ontology management software such as Protégé and Ontolingua. Among these, we have Chimaera (McGuinness, D. L. 2000), ODEMerge (Ramos, J. A., 2001) and Prompt (Noy, N. F. and Musen, M.

A., 2000). The ODEMerge tool performs its procedures automatically and uses a simple dictionary to identify synonyms and hypernym concepts.

The disadvantage of tools that perform the integration in an automatic manner is the inaccuracy of the mappings. As a matter of fact, sometimes they generate incorrect mappings. On the other hand, those that perform the integration interactively, also have a disadvantage, since they usually overwhelm the user with the verification of all the mappings found.

Another aspect related to schema integration consists of the fusion of data from different data sources into a consistent representation. Thus, it becomes necessary to solve the conflicts that arise from the different modes of representation of the same real world objects in multiple data sources. Bleiholder and Naumann (2006) describe and classify different strategies to handle inconsistent data, as follows:

- **Conflict ignoring –** consists of describing strategies that perform no decision whatsoever regarding conflicts. By employing this strategy, the system does not need to be aware of the conflicts in the data, as this information is not needed or used. These strategies are viable in any situation of integration, are easily implemented and have two representatives "Pass It On" and "Consider All Possibilities". In the "Pass It On" approach, all values are presented and the conflict resolution is deferred to the user. By contrast, the "Consider All Possibilities" approach tries to be the most complete possible, enumerating all eventualities and presenting the user with all possible combinations of attribute values and occasionally creates combinations that are not yet present in the sources.
- **Conflict avoiding –** these strategies acknowledge the existence of possible conflicts, but do not solve them.

In our work, the proposed management mechanism allows the user not only to obtain an ontology that represents the union of two ontologies, but it also supports the construction of an ontology that represents the intersection between two ontologies. The ***OntologyManagerTab*** application allows the user to save each resulting ontology so that it can be loaded and reworked as many times as it is necessary to achieve the desired result.

Table 3 presents a comparative analysis between the applications for ontology integration described in this chapter, as well as the ***OntologyManagerTab*** tool proposed in this dissertation.

Application	Chimaera	PROMP	ODEMerge	OntologyManagerTab
Matching Technique	Taxonomy and String based	Taxonomy, String and Graph based	Linguistic Resources based	String based
Ontology Language	Ontolingua, XOL	RDFS	RDFS, DAML + OIL	RDF(S)
Management Environment	Ontolingua	Protégé	WebODE	Protégé
Automation Level	Semi-automatic	Manual	Automatic	Automatic
Type of Integrated Elements	Classes and Properties	Classes, Properties and Instances	Classes and Properties	Classes, Properties and Restrictions
Conflict Handling Strategy	Conflict Ignoring	Conflict Ignoring	Conflict Ignoring	Conflict Avoiding

Table 3. Comparative analysis between the applications for ontology integration.

3.5 Summary

In this chapter, we presented work related to ontology management, separating them into areas of focus. We also explained why these areas are important to the ontology management process, addressing the existing approaches developed so far for each of them. We also enumerated what features our application, the ***OntologyManagerTab***, has to contribute to this process in each area.

Although ontology management strategies, proposed in the literature, use different approaches and processes, it is clear that none of them provide the user with an integrated tool for maintaining multiple ontologies, developing new ones, covering the areas of ontology reuse, ontology versioning and ontology-based data integration.

4
Constraint Graphs

4.1 Introduction

In this chapter, we will explain how to represent a finite set of lightweight constraints as a *constraint graph* (Casanova et al., 2011) that captures the structure of the logical implication. Constraint graphs lead to a procedure for checking inconsistencies in polynomial time relative to the size restrictions (Casanova et al., 2011).

We stress that the concepts introduced in this section refer only to lightweight inclusions. Therefore, we often omit explicit reference to this DL family in what follows, a simplification that the reader must bear in mind.

4.2 Constraint Graph Representation

Let $\boldsymbol{\Sigma}$ be a set of normalized lightweight constraints and $\boldsymbol{\Omega}$ a finite set of lightweight expressions, in other words, expressions that may occur in the right or left side of a normalized constraint (see Section 2.3). The *alphabet* of $\boldsymbol{\Sigma}$ and $\boldsymbol{\Omega}$ is defined as a finite set of atomic concepts and properties that occur in $\boldsymbol{\Sigma}$ and $\boldsymbol{\Omega}$.

We say that the *complement* of a basic concept description $\boldsymbol{b}$ is $\neg\boldsymbol{b}$, and vice-versa. If $\boldsymbol{e}$ is a basic concept description, or the negation of a basic concept description, then $\underline{\boldsymbol{e}}$ denotes the complement of $\boldsymbol{e}$ (Casanova *et al.*, 2012).

Definition 1:

A. 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.

(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 $\underline{\boldsymbol{e}}$.

(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 $(\geq n\ p)$ such that $m<n$, where p is either P or P^{-}, then there is an arc (N,M) in δ. Such arc are called *tautological arcs*.

(v) If there are nodes M and N in γ labeled with $\neg(\geq m\ p)$ and $\neg(\geq n\ p)$ such that $m<n$, where p is either P or P^{-}, then there is an arc (M,N) in δ. Such arc are called *tautological arcs*.

(vi) If there is an arc (M,N) in δ such that M and N are labeled with e and f, respectively, then there is an arc (K,L) in δ such that K and L are the nodes labeled with $\underline{\boldsymbol{f}}$ and $\underline{\boldsymbol{e}}$, respectively

(vii) These are the only nodes and arcs of $g(\Sigma,\Omega)$.

B. The *constraint graph* for Σ and Ω is the labeled graph $G(\Sigma,\Omega)=(\eta,\varepsilon,\lambda)$, where λ labels each node with a set of concept descriptions. The graph $G(\Sigma,\Omega)$ is defined by collapsing each strongly connected component of $g(\Sigma,\Omega)$ into a single node, labeled with the set of concept 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 graph that *captures* Σ. In what follows, we use $K \rightarrow M$ to indicate that there is a path in $G(\Sigma,\Omega)$ from K to M. In addition, as a convenience, a *path of length 0* is a path consisting of a single node. We now introduce the notion of constraint graph as follows.

Definition 2: Let $G(\Sigma,\Omega)=(\eta,\varepsilon,\lambda)$ be the constraint graph for Σ and Ω.

(i) We say that a node K of $G(\Sigma,\Omega)$ is a $\bot$-*node of rank 0* if

(a) K is labeled with $\bot$, or

(b) K is not labeled with $\bot$, there is no $\bot$-node L of rank 0 such that (K,L) is an arc of $G(\Sigma,\Omega)$, and there are nodes M and N, not necessarily distinct from K, and a basic concept description b such that

M and N are labeled with b and $\neg b$, respectively, and $K \rightarrow M$ and $K \rightarrow N$, or

(c) K is not labeled with $\bot$, there is no $\bot$-node L of rank 0 such that (K,L) is an arc of $G(\Sigma,\Omega)$, and there are nodes M and N, M is labeled with $\neg(\geq m\ p)$ and N is labeled with $(\geq m\ p)$, or M is labeled with $(\geq m\ p)$ and N is labeled with $\neg(\geq m\ p)$, where p is either P or P^{-}, and $K \rightarrow M$ and $K \rightarrow N$.

(ii) For a positive integer n, we say that a node K of $G(\Sigma,\Omega)$ is a $\bot$-*node of rank n* if K is not a $\bot$-node of rank m, with $m<n$, and there is a $\bot$-node L of rank $n-1$ such that

(a) (K,L) is an arc of $G(\Sigma,\Omega)$, or

(b) L is labeled with $(\geq 1\ P^{-})$ and K is labeled with $(\geq 1\ P)$, or

(c) L is labeled with $(\geq 1\ P)$ and K is labeled with $(\geq 1\ P^{-})$.

Case (ii-b) captures the fact that, given an interpretation $\boldsymbol{s}$, if $\boldsymbol{s}((\geq 1\ P^{-}))=\varnothing$, then $\boldsymbol{s}(P)=\boldsymbol{s}((\geq 1\ P))=\varnothing$. Case (ii-c) follows likewise, when $\boldsymbol{s}((\geq 1\ P))=\varnothing$. In view of these cases, the notion of rank is necessary to avoid a circular definition.

Definition 3: 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 rank n, for some non-negative integer n. We also say that K is a $\top$-*node* iff $\underline{K}$ is a $\bot$-*node*.

As a convenience, each node of a constraint graph will have Boolean tags corresponding to $\top$-*node* and $\bot$-*node,* and also an integer that marks its rank n in case of a $\bot$-*node.* These tags are completely independent from the node labels and are initialized as false when nodes are created.

4.3 Constraint Graph Basic Functions

The **OntologyManagerTab** tool uses constraint graphs as defined in Section 4.2. The generation of constraint graphs includes a normalization step, as discussed in Section 2.3. Furthermore, the normalization procedure uses a given base ontology to create a new ontology ignoring any elements that are not lightweight; this new ontology is then used to build the constraint graph.

Briefly, the construction of a constraint graph for a given ontology $O_1=(V_1,\Sigma_1)$ goes as follows:

1. Create a new ontology $O_2=(V_2,\Sigma_2)$ such that Σ_2 contains only the lightweight constraints of O_1 and V_2 contains only the symbols that occur in Σ_2.
2. Normalize the constraints in Σ_2, creating a new ontology $O_3=(V_3,\Sigma_3)$ such that $V_3=V_2$ and Σ_3 are the normalized versions of the constraints in Σ_2.
3. Create the constraint graph $G(\Sigma_3)$ for Σ_3 using **Definition 1**.
4. Tag $G(\Sigma_3)$ to indicate the $\top$-*node* and $\bot$-*node* using **Definition 2** and **Definition 3,** in this order.

The procedures that implement each of the operations, discussed in Section 5, return a set of constraints that may contain redundancies. Therefore, a graph minimization function is implemented to compute the minimal equivalent graph (MEG) of a constraint graph G. However, this function is not triggered automatically after every operation in case the user wishes to check the transitive closure obtained.

The MEG of a graph is defined as a graph H with a minimal set of edges such that the transitive closure of G and H are equal. This problem has a polynomial solution when G is acyclic and is NP-hard for strongly connected graphs (Aho, Garey and Ullman, 1972; Hsu, 1975; Khuller, Raghavachari and Young, 1975).

Figure 2 contains all basic procedures used in the graph construction, including the graph minimization procedure. These procedures will be referenced in the next section when describing the implementation of each operation.

SearchForInconsistencies:
Input: a tagged constraint graph $G(\Sigma_1,\Omega)$.

Output: a tagged constraint graph $G(\Sigma_2,\Omega)$.

(1) Initialize Σ_2 to be the empty set.
(2) Uses **Definition 2** over the set Σ_1 to populate Σ_2
(3) Uses **Definition 3** over Σ_2
(4) Return $G(\Sigma_2,\Omega)$.

LinkCardinalityRestrictions:

Input: a tagged constraint graph $G(\Sigma,\Omega)$.
Output: a tagged constraint graph $G(\Sigma,\Omega)$.

(1) If there are nodes M and N in G labeled with $(\geq m\ p)$ and $(\geq n\ p)$ such that $m<n$, where p is either P or P^{-}, then add an arc (N,M) to Σ
(2) If there are nodes M and N in G labeled with $\neg(\geq$m p) and $\neg(\geq$n p) such that m<n, where p is either P or P^{-}, then add an arc (M,N) to Σ
(3) $G(\Sigma,\Omega)$=**SearchForInconsistencies(**$G(\Sigma,\Omega)$**)**
(4) Return $G(\Sigma,\Omega)$.

ConstructGraph:

Input: a normalized lightweight ontology O.

Output: the tagged constraint graph $G(\Sigma,\Omega)$

(1) Construct the constraint graph $G(\Sigma,\Omega)$ for Σ and Ω , using **Definition 1**.
(2) $G(\Sigma_2,\Omega)$=**SearchForInconsistencies(**$G(\Sigma,\Omega)$**)**
(3) Return $G(\Sigma_2,\Omega)$.

MinimizeGraph:

Input: a tagged constraint graph G
Output: a MEG H of G

(1) Initialize H with the same nodes, arcs, labels and tags as G.
(2) For each node L of H labeled only with atomic concepts and at-least restrictions,
for each arc (L,M) in H,
for each node N in H, do:
if there are arcs (M,N) and (L,N) in H such that (L,N) is not a tautological arc, drop from H both the arc (L,N) and the arc $(\overline{N},\overline{L})$ connecting the dual nodes of L and M.

SaveOntology:

Input: a tagged constraint graph $G(\Sigma_1,\Omega)$
Output: an OWL file

(1) Initialize Σ_2 to be the empty set.
(2) Mark all arcs of H as unprocessed.
(3) For each node M of H labeled only with atomic concepts and at-least restrictions, do:
(4) If M is tagged as a "$\perp$-node", then
(5) For each label e of M,
(6) Add to Σ_2 a constraint of the form $e \sqsubseteq \perp$.
(7) If M is not tagged as "$\perp$-node", then
(8) Order the labels of M, creating a list $e_1,\ldots,e_n$, and
(9) Add to Σ_2 the constraints $e_1 \sqsubseteq e_2$, $e_2 \sqsubseteq e_3$,…, $e_{n-1} \sqsubseteq e_n$ and $e_n \sqsubseteq e_1$.
(10) For each arc (M,N) of H such that (M,N) is unprocessed, do:
(11) Select a label e of M and a label f of N and
(12) Add to Σ_2 a constraint of the form $e \sqsubseteq f$.
(13) Mark both (M,N) and $(\overline{N},\overline{M})$ as processed.
(14) Composes G as $G(\Sigma_2,\Omega)$
(15) For each node L of G labeled only with atomic concepts and at-least restrictions,
(16) If the node is not a complement of class or complement of property
(17) Save its description and if there are arcs (L,N)
(18) For each node N in G, do:

```
(19)    Obtain the description of N and save a constraint (L,N) to the file
(20)    using the description of both nodes
```

Figure 2. Basic Procedures

Regarding the complexity of the operations in **Figure 2** we can evaluate each one in terms of the number of nodes n and the number of edges m of the graph G. **SearchForInconsistencies** searches G for bottom nodes, starting from each node of G; therefore it has complexity $O(n(n+m))$. Similarly, **LinkCardinalityRestrictions** and **ConstructGraph** have complexity $O(n(n+m))$. **MinimizeGraph** is implemented with complexity $O(mn^2)$. Finally, **SaveOntology** goes through the graph of the resulting ontology saving its terms and constraints and is implemented with complexity $O(n+m)$.

4.4 Summary

In this chapter, we presented the definition of the constraint graph $G(\Sigma,\Omega)$ with its specification and implementation. We also included basic procedures to manipulate this graph, which will be referenced in the specification of the proposed operations in the next chapter.

5
Implementation of the Operations

5.1 Introduction

In this chapter, we will first define the operations over lightweight ontologies considered in this dissertation: **Projection, Union, Intersection** and **Difference**. Then, we will describe the algorithms that implement each operation; all based on the notion of constraint graph introduced in Chapter 4.

This chapter is structured as follows. Section 5.2 will define the operations. Then, Sections 5.3 to 5.6 describe the algorithms that implement each operation. Section 5.7 contains the final considerations.

5.2 Definition of Operations over Lightweight Ontologies

Recall that an ontology is a pair $\boldsymbol{O}=(V,\Sigma)$ such that V is a finite alphabet, called the *vocabulary* of $\boldsymbol{O}$, whose atomic concepts and atomic roles are called *classes* and *properties* of $\boldsymbol{O}$, respectively, and Σ is a set of inclusions in V, called the *constraints* of $\boldsymbol{O}$. In particular, *lightweight ontologies* are ontologies whose constraints are lightweight inclusions (see Table 1 on Chapter 2). Also, recall that the *theory* of Σ in V, denoted $\tau[\Sigma]$, is the set of all inclusions in V that are logical consequence of Σ.

Definition 4 introduces the operations over lightweight ontologies. 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.

Definition 4: Let $O_1 = (V_1, \Sigma_1)$ and $O_2 = (V_2, \Sigma_2)$ be two lightweight ontologies, W a subset of V_1, Σ_1 is the constraint set over V_1 and Σ_2 is the constraint set over V_2.

(i) 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 a set of constraints in $\Sigma_P = \tau[\Sigma_1]$ that uses only symbols from W.

(ii) 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 $\Sigma_U = \Sigma_1 \cup \Sigma_2$.

(iii) 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 $O_{Int}=(V_{Int},\Sigma_{Int})$, where $V_{Int} = V_1 \cap V_2$ and $\Sigma_{Int} = \tau[\Sigma_1] \cap \tau[\Sigma_2]$.

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

Also, when comparing distinct constraint graphs, the equivalent nodes and constraints between these graphs need to be mapped; these are discovered according to **Definition 5.**

Definition 5: Let $G_1(V_1,\Sigma_1)$ and $G_2(V_2,\Sigma_2)$ be two distinct constraint graphs.

(i) We say that a node K of $G_1(V_1,\Sigma_1)$ is *equivalent* to a node M of $G_2(V_2,\Sigma_2)$ iff K is tagged with the same terms as M. In case of cardinality constraints, the *equivalency* treats the cardinality value differently according to which operation is being processed.

(ii) We say that a constraint C of $G_1(V_1,\Sigma_1)$ is *equivalent* to a constraint D of $G_2(V_2,\Sigma_2)$ iff the nodes K and L related by C are *equivalent* to the nodes Q and P related by D, respectively.

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 (Hepp M. *et al*., 2008). 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.

5.3 Implementation of Projection

Let $O_1 = (V_1, \Sigma_1)$ be a lightweight ontology and W be a subset of V_1. The **Projection** procedure computes Γ_P so that $\tau[\Gamma_P] = \tau[\Sigma_P]$. That is, given any lightweight inclusion $e \sqsubseteq f$ that involves only classes and properties in W, $e \sqsubseteq f$ is a logical

consequence of Γ_P iff $e \sqsubseteq f$ is a logical consequence of Σ_1. Note that this does not mean that $e \sqsubseteq f$ is a logical consequence of the subset of Σ_1 whose inclusions involve only classes and properties in W.

Projection procedure:

---

1. **Construct** $G(V_1,\Sigma_1)$, the normalized tagged constraint graph for $O_1=(V_1,\Sigma_1)$.
2. **Construct** $G^*(V_1,\Sigma_1^*)$, the transitive closure of $G(V_1,\Sigma_1)$. The nodes of $G^*(V_1,\Sigma_1^*)$ retain all labels and tags as in $G(V_1,\Sigma_1)$.
3. Use $G^*(V_1,\Sigma_1^*)$ to create a graph G_W by discarding all concept descriptions that label nodes of $G^*(V_1,\Sigma_1^*)$ and that involve classes and properties which are not in W; nodes that end up with no labels are discarded, as well as their adjacent arcs. The nodes of G_W retain all tags as in $G^*(V_1,\Sigma_1^*)$.
5. Call **LinkCardinalityRestrictions** with G_W to generate G_{W1}.
6. Call **SaveOntology** with G_{W1} to generate Γ_P.
7. **Return** $O_P = (W, \Gamma_P)$.

Example 1 illustrates how **Projection** operates.

Let $O_1 = (V_1,\Sigma_1)$ and $O_2 = (V_2,\Sigma_2)$ be lightweight ontologies. The **Union** procedure returns the ontology $O_U = (V_U, \Sigma_U)$, where $V_U = V_1 \cup V_2$ and $\Sigma_U = \Sigma_1 \cup \Sigma_2$. The **Union** procedure uses a node list $L_e = V_1 \cap V_2$.

Example 1: Consider the following vocabulary:

V_{FF} = {foaf:Agent, foaf:Person, foaf:Organization, foaf:account }

The projection of the *FOAF* ontology over V_{FF} is the *Foaf Facebook* ontology, $\boldsymbol{FF} = (V_{FF},\Sigma_{FF})$, where Σ_{FF} is the set of constraints shown in Table 4.

	Constraint	Informal specification
1.	foaf:Person ⊑ foaf:Agent foaf:Organization ⊑ foaf:Agent	foaf:Person is a subset of foaf:Agent foaf:Organization is a subset of foaf:Agent
2.	(≥1 foaf:account) ⊑ foaf:Agent	The range of foaf:account is foaf:Agent
3.	foaf:Person ⊑ ¬foaf:Organization	G(Σ_{FOAF}) has a path from the node labeled with foaf:Person to the node labeled with ¬foaf:Organization, which indicates that foaf:Organization and

		foaf:Person are disjoint

Table 4. Constraints of Ontology FF

The complexity of this operation can be estimated as $O(n^3)$ regarding the original ontology O_1. The key factor to achieve this processing time is the use of the *Transitive Closure*, which in this software is implemented with such complexity.

5.4 Implementation of Union

Let $O_1 = (V_1,\Sigma_1)$ and $O_2 = (V_2,\Sigma_2)$ be lightweight ontologies. The **Union** procedure returns the ontology $O_U = (V_U, \Sigma_U)$, where $V_U = V_1 \cup V_2$ and $\Sigma_U = \Sigma_1 \cup \Sigma_2$. The **Union** procedure uses a node list $L_e = V_1 \cap V_2$.

Union procedure:

---

1. **Construct** $G_1(V_1,\Sigma_1)$, the normalized tagged constraint graph for $O_1=(V_1,\Sigma_1)$.
2. **Construct** $G_2(V_2,\Sigma_2)$, the normalized tagged constraint graph for $O_2=(V_2,\Sigma_2)$.
3. **Initialize** G_U as a copy of G_1, which gives us $G_U(V_1,\Sigma_1)$.
4. **For each** node ***e*** in V_2, search G_U for an **equivalent** node:
   - If the node ***e*** is equivalent to the node ***f*** in V_U, add the reference ***e, f*** to L_e.
   - Otherwise add ***e*** to V_U, adding the reference between the node ***e*** in V_2 and the new *node* ***n*** in V_U to L_e.
5. **For each** constraint ***c*** in Σ_2, search G_U for an **equivalent** constraint:
   - If the constraint ***c*** is not equivalent to any constraint in V_U, add ***c*** to Σ_U, remembering to query L_e for the equivalent nodes when adding the constraint.
6. Call **LinkCardinalityRestrictions** with G_U to generate G_{U1}.
7. Call **SaveOntology** with G_{U1} to generate Γ_U.
8. **Return** $O_U = (V_U , \Gamma_U)$.

Example 2 illustrates how **Union** operates.

Example 2: Consider the following ontologies:

$O_1 = (V_1, \Sigma_1)$ with:

V_1 = {foaf:Agent, foaf:homepage}

Σ_1 = {(≥1 foaf:homepage) ⊑ foaf:Agent}

$O_2 = (V_2, \Sigma_2)$ with:

V_2 = {foaf:Agent, foaf:homepage}

Σ_2 = {(≥10 foaf:homepage) ⊑ foaf:Agent}

The union of O_1 and O_2 is the ontology $O_U = (V_U, \Sigma_U)$, where $V_U = V_1 = V_2$ and Σ_U is the set of constraints shown in **Table 5**.

If the user chooses to minimize the graph G_U that corresponds to the ontology O_U, constraints 2 and 3 from **Table 5** will be eliminated, leaving O_U with the same terms and constraints as O_1.

	Constraint	Informal specification
1.	(≥1 foaf:homepage) ⊑ foaf:Agent	As in O_1, the range of foaf:homepage is foaf:Agent
2.	(≥10 foaf:homepage) ⊑ foaf:Agent	As in O_2, the range of foaf:account is foaf:Agent with minimum cardinality 10
3.	(≥10 foaf:homepage) ⊑ (≥1 foaf:homepage)	The resulting ontology O_3 generates this constraint.

Table 5. Constraints Σ_U of Ontology O_U.

The **Union** procedure also tests if the resulting ontology has constraints that force a class to be always empty (that is, equivalent to the bottom concept). **Example 3** illustrates this case.

Example 3: Consider the following ontologies of two distinct phone companies, both based on a simpler and older ontology:

$O_{PhoneCompany1} = (V_{PhoneCompany1}, \Sigma_{PhoneCompany1})$ with:

$V_{PhoneCompany1}$ = {pc:Phone, pc:MobilePhone, pc:FixedPhone, pc:Call, pc:MobileCall}

$\Sigma_{PhoneCompany1}$ = {pc:MobilePhone ⊑ pc:Phone,

pc:FixedPhone ⊑ pc:Phone, pc:MobilePhone ⊑ pc:FixedPhone,
pc:FixedPhone ⊑ ¬pc:MobilePhone, pc:Call ⊑ ¬pc:MobileCall,
pc:MobileCall ⊑ ¬pc:Call }

$O_{PhoneCompany2} = (V_{PhoneCompany2}, \Sigma_{PhoneCompany2})$ with:

$V_{PhoneCompany2}$ = {pc:Phone, pc:MobilePhone, pc:FixedPhone, pc:Call, pc:MobileCall }

$\Sigma_{PhoneCompany2}$ = { pc:MobilePhone ⊑ pc:Phone,
pc:FixedPhone ⊑ pc:Phone, pc:MobileCall ⊑ pc:Call }

The **Union** of $O_{PhoneCompany1}$ and $O_{PhoneCompany2}$ is the ontology $O_{PhoneCompany3}$, which has the same vocabulary as the original ontologies and the set of constraints Σ_U shown in **Table 6**. We note that **Table 6** shows the constraints as they are computed by our application, where the expression in the first column is contained (⊑) in the expression that appears in the second column.

However, $O_{PhoneCompany3}$ has a class which is always empty since classes pc:Call and pc:MobileCall are disjoint in $O_{PhoneCompany1}$ and class pc:MobileCall is a subset of class pc:Call in $O_{PhoneCompany2}$. This may typically happen when ontologies are separately developed using the same ontology as origin.

	(a) $O_{PhoneCompany1}$		(b) $O_{PhoneCompany2}$		(c) $O_{PhoneCompany3}$	
1	Mobile Phone	Phone	MobilePhone	Phone	MobilePhone	Phone
2	FixedPhone	Phone	FixedPhone	Phone	FixedPhone	Phone
3	MobilePhone	¬ FixedPhone	MobileCall	Call	MobilePhone	¬ FixedPhone
5	FixedPhone	¬ MobilePhone			FixedPhone	¬ MobilePhone
6	Call	¬ MobileCall			Call	¬ MobileCall
7	MobileCall	¬ Call			MobileCall	⊥

Table 6. Constraints for $O_{PhoneCompany1}$, $O_{PhoneCompany2}$ and their Union $O_{PhoneCompany3}$

The complexity of this operation can be estimated as $O(n^3)$ regarding the resulting ontology O_U. The key factor to achieve this processing time is the use of the **LinkCardinalityRestrictions**, which in this software is implemented with such complexity as shown in Section 4.3.

5.5 Implementation of Intersection

Let $O_1 = (V_1,\Sigma_1)$ and $O_2 = (V_2,\Sigma_2)$ be lightweight ontologies. The **Intersection** procedure returns the ontology $O_{Int} = (V_{Int}, \Sigma_{Int})$, where $V_{Int} = V_1 \cap V_2$ and $\Sigma_{Int} = \tau[\Sigma_1] \cap \tau[\Sigma_2]$. As the **Union** procedure, **Intersection** also uses a node list $L_e = V_1 \cap V_2$, which will help compute $\Sigma_{Int} = \tau[\Sigma_1] \cap \tau[\Sigma_2]$.

The **Intersection** procedure

---

1. **Construct** $G_1(V_1,\Sigma_1)$, the normalized tagged constraint graph for $O_1=(V_1,\Sigma_1)$.
2. **Construct** $G_2(V_2,\Sigma_2)$, the normalized tagged constraint graph for $O_2=(V_2,\Sigma_2)$.
3. **Construct** $G_1^*(V_1,\Sigma_1^*)$, the transitive closure of $G_1(V_1,\Sigma_1)$. The nodes of $G_1^*(V_1,\Sigma_1^*)$ retain all labels and tags as in $G_1(V_1,\Sigma_1)$.
5. **Construct** $G_2^*(V_2,\Sigma_2^*)$, the transitive closure of $G_2(V_2,\Sigma_2)$. The nodes of $G_2^*(V_2,\Sigma_2^*)$ retain all labels and tags as in $G_2(V_2,\Sigma_2)$.
6. **Initialize** $G_{Int}\ (V_{Int}, \Sigma_{Int})$ as an empty graph.
4. **For each** node ***e*** in V_1, search G_2^* for an **equivalent** node:
   - If the node ***e*** is equivalent to the node ***f*** in V_2, add the node to G_{Int} and add the reference ***e, f, n*** to L_e, where ***n*** the new node in V_{Int}.
5. **For each** constraint ***c*** in Σ_1^*, search Σ_2^* for an **equivalent** constraint:
   - If the constraint ***c*** is equivalent to the constraint ***d*** in Σ_2^* add ***c*** to Σ_{Int}, remembering to query L_e for the equivalent terms when adding the constraint.
6. Call **LinkCardinalityRestrictions** with G_{Int} to generate G_{Int1}.
7. Call **SaveOntology** with G_{Int1} to generate Γ_{Int}.
8. **Return** $O_{Int} = (V_{Int}, \Gamma_{Int})$.

Example 4 illustrates how **Intersection** operates.

Example 4: Consider the following ontologies:

$O_1 = (V_1,\Sigma_1)$ with:

V_1 = {foaf:Agent, foaf:homepage}

Σ_1 = {(≥1 foaf:homepage) ⊑ foaf:Agent}

$O_2 = (V_2, \Sigma_2)$ with:

V_2 = {foaf:Agent, foaf:homepage}

Σ_2 = {(≥10 foaf:homepage) ⊑ foaf:Agent}

The intersection of O_1 and O_2 is the ontology $O_{int} = (V_{int}, \Sigma_{int})$, where $V_{int} = V_1 = V_2$ and Σ_{int} is the set of constraints shown in **Table 7**.

	Constraint	Informal specification
1.	(≥10 foaf:homepage) ⊑ foaf:Agent	As in O_2, the range of foaf:homepage is foaf:Agent with minimum cardinality 10

Table 7. Constraints Σ_{int} of Ontology O_{int}

Example 5 illustrates a case analogous to that discussed in **Example 3**.

Example 5: Consider the following ontologies of two distinct phone companies, both based on a simpler and older ontology:

$O_{PhoneCompany1} = (V_{PhoneCompany1}, \Sigma_{PhoneCompany1})$ with:

$V_{PhoneCompany1}$ = {pc:Phone, pc:MobilePhone, pc:FixedPhone, pc:Call, pc:MobileCall}

$\Sigma_{PhoneCompany1}$ = {pc:MobilePhone ⊑ pc:Phone, pc:FixedPhone ⊑ pc:Phone, pc:MobilePhone ⊑ pc:FixedPhone, pc:FixedPhone ⊑ ¬pc:MobilePhone, pc:Call ⊑ ¬pc:MobileCall, pc:MobileCall ⊑ ¬pc:Call }

$O_{PhoneCompany2} = (V_{PhoneCompany2}, \Sigma_{PhoneCompany2})$ with:

$V_{PhoneCompany2}$ = {pc:Phone, pc:MobilePhone, pc:FixedPhone, pc:Call, pc:MobileCall }

$\Sigma_{PhoneCompany2}$ = { pc:MobilePhone ⊑ pc:Phone, pc:FixedPhone ⊑ pc:Phone, pc:MobileCall ⊑ pc:Call }

The **Intersection** of $O_{PhoneCompany1}$ and $O_{PhoneCompany2}$ is the ontology $O_{PhoneCompany4}$, which has the same vocabulary as the original ontologies and the set of constraints Σ_U shown in **Table 8**. Again, we note that **Table 8** shows the constraints as they are computed by our application, where the expression in the first column is contained (⊑) in the expression that appears in the second column.

(a) $O_{PhoneCompany1}$	(b) $O_{PhoneCompany2}$	(c) $O_{PhoneCompany3}$

1	MobilePhone	Phone		MobilePhone	Phone	MobilePhone	Phone
2	FixedPhone	Phone		FixedPhone	Phone	FixedPhone	Phone
3	MobilePhone	¬ FixedPhone		MobileCall	Call		
5	FixedPhone	¬ MobilePhone					
6	Call	¬ MobileCall					
7	MobileCall	¬ Call					

Table 8. Constraints for $O_{PhoneCompany1}$, $O_{PhoneCompany2}$ and their Intersection $O_{PhoneCompany3}$

The complexity of this operation can be estimated as $O(n^3)$ regarding the original ontology with greater number of nodes and edges, O_1 or O_2. The key factor to achieve this processing time is the use of the *Transitive Closure*, which in this software is implemented with such complexity.

5.6 Implementation of Difference

Let $O_1 = (V_1,\Sigma_1)$ and $O_2 = (V_2,\Sigma_2)$ be lightweight ontologies that represent two versions for the same ontology. The **Difference** of $O_1 = (V_1, \Sigma_1)$ and $O_2 = (V_2, \Sigma_2)$, represented by $O_1 - O_2$, returns the ontology $O_D = (V_D, \Sigma_D)$, where $V_D = V_1 - V_2$ and $\Sigma_D = \tau[\Sigma_1] - \tau[\Sigma_2]$.

The problem of creating a procedure to compute the difference between two ontologies, $O_1=(V_1,\Sigma_1)$ and $O_2=(V_2,\Sigma_2)$, lies in the fact that it might not be possible to obtain a finite set of inclusions Δ_N in such a way that

(1) $\tau[\Delta_N] = \tau[\Sigma_1] - \tau[\Sigma_2]$

This invalidates the effort to create a procedure to obtain a finite set of inclusions Δ_N satisfying (1), along the lines of those exhibited in Sections 5.2 and 5.3. This remark in fact puts in doubt the usefulness of a (generic) difference operation.

For example, consider the following two sets of inclusions:

(2) $\Sigma_1 = \{ e \sqsubseteq g, g \sqsubseteq f \}$

(3) $\Sigma_2 = \{ e \sqsubseteq f \}$

Then, ignoring tautologies when computing $\tau[\Sigma_j], j=1,2$, we have:

(4) $\tau[\Sigma_1] = \{ e \sqsubseteq g, g \sqsubseteq f, e \sqsubseteq f \}$

(5) $\tau[\Sigma_2] = \{ e \sqsubseteq f \}$

(6) $\Delta_N = \tau[\Sigma_1] - \tau[\Sigma_2] = \{ e \sqsubseteq g, g \sqsubseteq f \} = \Sigma_1$

But this definition of Δ_N is not satisfactory, since we have

(7) $\tau[\Delta_N] = \tau[\Sigma_1] = \{ e \sqsubseteq g, g \sqsubseteq f, e \sqsubseteq f \}$

That is, to compute the difference $\Delta_N = \tau[\Sigma_1] - \tau[\Sigma_2]$, we remove "$e \sqsubseteq f$" from $\tau[\Sigma_1]$, only to get "$e \sqsubseteq f$" back by logical implication from Δ_N. In fact, in this rather obvious example, we cannot obtain a set of inclusions Δ_N such that $\tau[\Delta_N] = \tau[\Sigma_1] - \tau[\Sigma_2]$. Indeed, since set of inclusions must not logically imply "$e \sqsubseteq f$", the only candidates are:

(8) $\Delta_1 = \{ e \sqsubseteq g \}$

(9) $\Delta_2 = \{ g \sqsubseteq f \}$

In both cases, we have that (again ignoring tautologies when computing $\tau[\Delta_k]$, $k=1,2$): $\tau[\Delta_k] = \Delta_k \subset \tau[\Sigma_1] - \tau[\Sigma_2]$

The **Difference** procedure

---

1. **Construct** $G_1(V_1,\Sigma_1)$, the normalized tagged constraint graph for $O_1=(V_1,\Sigma_1)$.
2. **Construct** $G_2(V_2,\Sigma_2)$, the normalized tagged constraint graph for $O_2=(V_2,\Sigma_2)$.
3. **Construct** $G_1^*(V_1,\Sigma_1^*)$, the transitive closure of $G_1(V_1,\Sigma_1)$. The nodes of $G_1^*(V_1,\Sigma_1^*)$ retain all labels and tags as in $G_1(V_1,\Sigma_1)$.
4. **Construct** $G_2^*(V_2,\Sigma_2^*)$, the transitive closure of $G_2(V_2,\Sigma_2)$. The nodes of $G_2^*(V_2,\Sigma_2^*)$ retain all labels and tags as in $G_2(V_2,\Sigma_2)$.
5. **Initialize** $G_D\ (V_D\ ,\ \Sigma_D)$ as an empty graph.
6. **For each** node ***e*** in V_1, search G_2^* for an **equivalent** node:
   - If the node ***e*** is not equivalent to any node in V_2, add the node to G_D with its constraints from Σ_1^*.
7. **For each** constraint ***c*** in Σ_1^*, search Σ_2^* for an **equivalent** constraint:
   - If the constraint ***c*** is not equivalent to any constraint in Σ_2^*, add ***c*** to Σ_D, adding the nodes in the constraint to G_D, if they are not already in V_D.
8. Call **LinkCardinalityRestrictions** with G_D to generate G_{D1}.
9. Call **SaveOntology** with G_{D1} to generate Γ_D.

10. **Return** $O_D = (V_D , \Gamma_D)$.

Examples 6 and **7** illustrate how **Difference** operates.

Example 6: Consider the following ontologies:

$O_1 = (V_1, \Sigma_1)$ with:

$V_1 = \{ A, B, C, D \}$

$\Sigma_1 = \{ A \sqsubseteq B , B \sqsubseteq C, C \sqsubseteq D \}$

$O_2 = (V_2, \Sigma_2)$ with:

$V_2 = \{ A, B, C \}$

$\Sigma_2 = \{ A \sqsubseteq B , B \sqsubseteq C \}$

The **Difference** between O_1 and O_2, is the ontology O_D, which has the vocabulary $V_D = V_1$ and the set of constraints Σ_D shown in **Table 9**.

Constraint		Informal specification
1.	$A \sqsubseteq D$	As in O_1, A is a subset of D and this inclusion is not in $\tau[\Sigma_2]$
2.	$B \sqsubseteq D$	As in O_1, B is a subset of D and this inclusion is not in $\tau[\Sigma_2]$
3.	$C \sqsubseteq D$	As in O_1, C is a subset of D and this inclusion is not in $\tau[\Sigma_2]$

Table 9. Constraints Σ_D of Ontology O_D

Example 7: Consider the following ontologies of two distinct phone companies, both based on a simpler and older ontology:

$O_{PhoneCompany1} = (V_{PhoneCompany1}, \Sigma_{PhoneCompany1})$ with:

$V_{PhoneCompany1}$ = {pc:Phone, pc:MobilePhone, pc:FixedPhone, pc:Call, pc:MobileCall}

$\Sigma_{PhoneCompany1}$ = {pc:MobilePhone ⊑ pc:Phone, pc:FixedPhone ⊑ pc:Phone, pc:MobilePhone ⊑ pc:FixedPhone, pc:FixedPhone ⊑ ¬pc:MobilePhone, pc:Call ⊑ ¬pc:MobileCall, pc:MobileCall ⊑ ¬pc:Call }

$O_{PhoneCompany2} = (V_{PhoneCompany2}, \Sigma_{PhoneCompany2})$ with:

$V_{PhoneCompany2}$ = {pc:Phone, pc:MobilePhone, pc:FixedPhone, pc:Call, pc:MobileCall }

$\Sigma_{PhoneCompany2}$ = { pc:MobilePhone ⊑ pc:Phone,

pc:FixedPhone ⊑ pc:Phone, pc:MobileCall ⊑ pc:Call }

The **Difference** between $O_{PhoneCompany1}$ and $O_{PhoneCompany2}$ is the ontology $O_{PhoneCompany5}$, which has the same vocabulary as $O_{PhoneCompany1}$ and the set of constraints Σ_D shown in **Table 10**.

	(a) $O_{PhoneCompany1}$		(b) $O_{PhoneCompany2}$		(c) $O_{PhoneCompany3}$	
1	MobilePhone	Phone	MobilePhone	Phone	MobilePhone	¬ FixedPhone
2	FixedPhone	Phone	FixedPhone	Phone	FixedPhone	¬ MobilePhone
3	MobilePhone	¬ FixedPhone	MobileCall	Call	Call	¬ MobileCall
5	FixedPhone	¬ MobilePhone			MobileCall	¬ Call
6	Call	¬ MobileCall				
7	MobileCall	¬ Call				

Table 10. Constraints for $O_{PhoneCompany1}$, $O_{PhoneCompany2}$ and their Difference $O_{PhoneCompany3}$

The complexity of this operation can also be estimated as $O(n^3)$ regarding the original ontology with greater number of nodes and edges, O_1 or O_2 . The key factor to achieve this processing time is the use of the *Transitive Closure*, which in this software is implemented with such complexity.

5.7 Summary

In this chapter, we first defined operations over lightweight ontologies and then presented the procedures that compute the operations. The application described in Chapter 6 implements all such procedures, with the intention of assisting the domain specialist in choosing or creating a new ontology to publish data on the Web.

Projection allows the domain specialist to retrieve only the fragment of an ontology needed for his application. **Union** merges two different ontologies. **Intersection** compares two ontologies and retrieves only the common terms and axioms. **Difference** compares two ontologies and retrieves only the axioms that are contained in the first, but not in the second.

These operations are meant to encourage and help the user follow the *Linked Data Principles*, allowing him to obtain new consistent ontologies using

others as base, thereby automating the onerous task of creating ontologies from scratch.

6
OntologyManagerTab – an Ontology Manager Plug-in for Protégé

6.1 Introduction

There are a wide variety of tools available, with different approaches and processes, which provide ontology management. However, few tools assist the domain specialist in the development of an ontology that represents a correct understanding of the semantics of the involved ontologies, since this requires taking into account not only the terms from the original ontologies but also their logical constraints. Furthermore, the use of several tools during the process of managing ontologies increases the manual work that must be performed by the domain specialist, making it even more onerous.

The ***OntologyManagerTab***, presented in this chapter, offers the ontology operations described in Chapter 5, integrated with traditional ontology management features. ***OntologyManagerTab*** was developed in Java as a tab plug-in over Protégé 3.4.8 (the implementation might require minor modifications to work with other versions of Protégé).

Despite the fact that ***OntologyManagerTab*** was developed as a Protégé plug-in, it 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 easier to adapt as a plug-in for any other framework or as a stand-alone software.

This chapter provides an overview of the developed software and is structured in the following manner. Section 6.2 gives a brief overview of the classes in the software and how they interact. Section 6.3 explains how to compile and setup ***OntologyManagerTab***. Section 6.4 explains, with the help of examples, how to use each operation implemented in the ***OntologyManagerTab***. Finally, Section

6.5 discusses the ontologies used in the experiments and the complexity of the operations.

6.2 Class Architecture

In this section, a brief overview of the class architecture of ***OntologyManagerTab*** *is presented.* The software source code is divided into three packages, containing a total of sixteen classes and two enumeration interfaces. The package specifications are:

1. Main – this package contains the OntologyManagerTab class, which is the main class in the project, that implements all operations discussed in Section 5 and integrates the software with the Protégé framework.
2. Application – this package contains classes that implement the Java interfaces required for the application to run.
3. Ontology – this package contains classes that implement the constraint graph, including the normalization and creation procedures.

Figure 3 shows how the packages interact. The *Main* package contains a single class, the OntologyManagerTab class, which creates the interface with the Protégé framework, sets the software as a tab in the program, treats the GUI events sent by the user and manipulates the classes from the *Application* and *Ontology* packages to run the required procedures. In addition, **Projection, Union, Intersection** and **Difference** operations are all implemented to run in separate threads from the user interface in order not to freeze in case of longer computations.

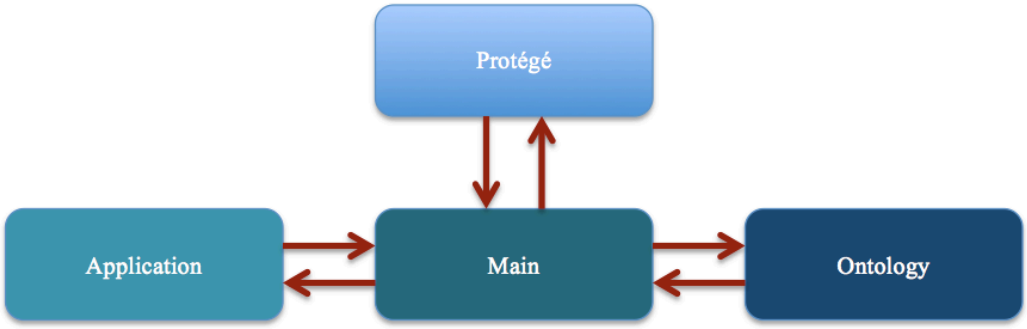

Figure 3. Packages Interactions

The *Application* package contains one enumeration interface and two classes:

- The Enumeration Interface *OSType* – classifies the Operating System that is running our software between a Windows, Linux or Mac type. This is extremely important since our tool opens and saves ontology files and must know which type of file system is being used.
- The ProjectionTableModel Class – extends the AbstractTableModel Java class to implement a different sort of table for the **Projection** operation, including among other things the checkbox used by the GUI in the procedure.
- The ProjectionItem Class – implements the objects that are used in each row of the ProjectionTableModel class and contain the details of the possible nodes for the **Projection** operation.

The *Ontology* package contains one enumeration interface and fourteen classes:

- The UsefulOWL Class – is an auxiliary class that contains small functions to handle OWL elements, includes mostly functions for the search and extraction of strings from the OWL elements that are widely used throughout the software.
- The Normalization Class – implements the procedures to normalize and extract the lightweight constraints from an OWL ontology, according to the rules defined in Table 1 in Section 2.5, saving this new normalized ontology in the same folder as the original.
- The Graph Class – implements a graph representation for an OWL ontology; it contains a list of nodes to represent the atomic concepts, properties and cardinality restrictions, as well as an adjacency list to represent the ontology constraints.
- The Graph.Edge Class – is a class contained in the graph class that implements edges between graph nodes, these edges are obtained from the adjacency list in the graph class.

- The Node Class – is the basic class of graph node, that is extended to characterize each specific type of object found in an OWL ontology.
- The NodeClass Class – specificies the class Node to represent an OWL Class Object.
- The NodeProperty Class – specificies the class Node to represent an OWL Property Object.
- The NodeRestrictionCardinality Class – specifies the class Node to represent an OWL Minimum Cardinality Object.
- The NodeRestrictionComplementOfClass Class – specifies the class Node to represent the complement of an OWL Class Object.
- The NodeRestrictionComplementOfProperty Class – specifies the class Node to represent the complement of an OWL Property Object.
- The NodeRestrictionComplementOfRestrictionCardinality Class – specifies the class Node to represent an OWL Maximum Cardinality Object.
- The Enumeration Interface NodeType – classifies the Node class as NodeClass, NodeProperty, NodeRestrictionCardinality, NodeRestrictionComplementOfClass, NodeRestrictionComplementOfProperty and NodeRestrictionComplementOfRestrictionCardinality.
- The ConstraintGraph Class – implements the procedures described in **Figure 2**, except for the SaveOntology procedure, to create the constraint graph, over which all the operations are realized.
- The BreadthFirstSearch Class – implements a breadth first search algorithm over the Graph class and also a transitive closure.
- The SaveOntology Class – implements the SaveOntology procedure from **Figure 2**.

The Graph and Node classes, as well as their specializations, are used to store the OWL ontologies over which the **Projection, Union, Intersection** and **Difference** operations are executed. The classes from the *Ontology* package interact with the *Main* package as shown in **Figure** 4.

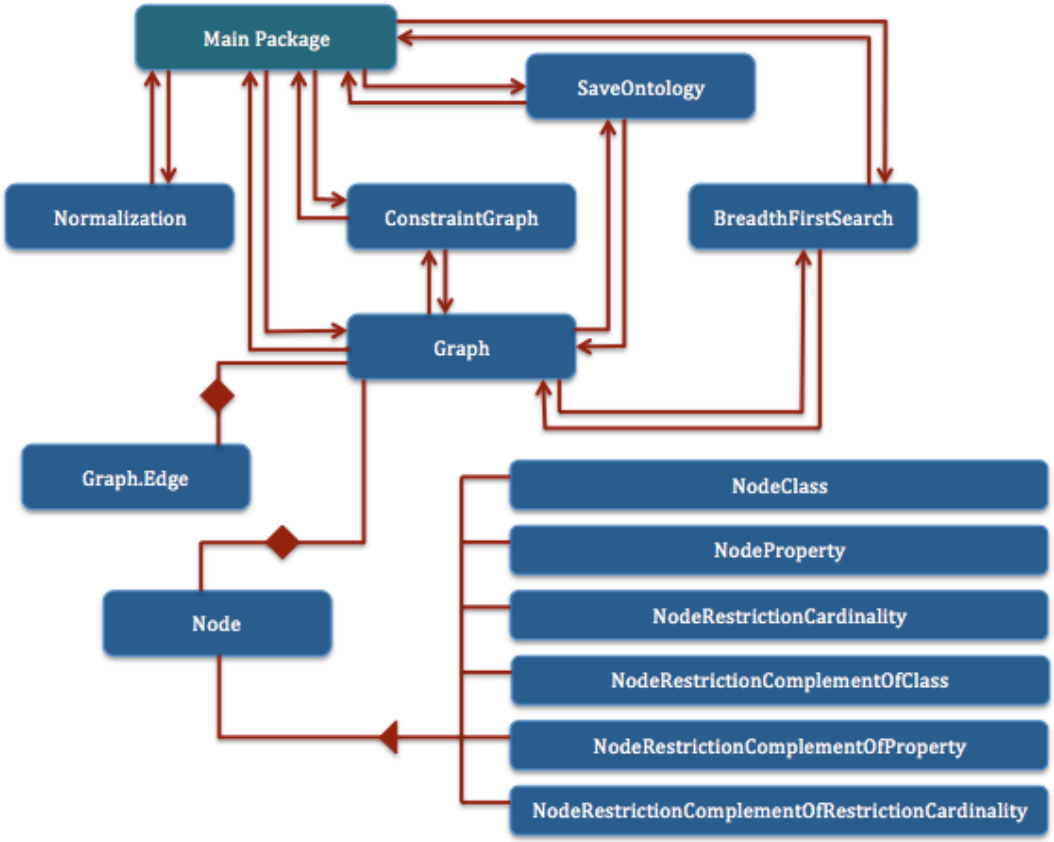

Figure 4. Ontology Package interaction with Main Package

6.3 Software Setup

The ***OntologyManagerTab*** software was developed entirely in Java using the Eclipse IDE. It has been implemented and tested with Protégé 3.4.8, running in Mac OSX 10.9. Although it has been only tested with this environment, the ***OntologyManagerTab*** was developed to run in any operating system as long as it supports the Protégé framework. To execute the project, the following steps must be followed:

1. Install Protégé 3.4.8.
2. Import the Eclipse Project.
3. In the Project Properties inside Eclipse, follow the path "Java Build Path ➔ Libraries" and add the three "External JARs" that accompany the project: looks.jar, protege.jar, owlapi-distribution-3.5.0.jar.
4. Compile and run the project.

The external JARs looks.jar and protege.jar provided in the project are specifically for Protégé version 3.4.8. To use any other version of the Protégé framework, both of these external JARs should be imported from the Protégé installation folder.

After compiling and running the project, Protégé will be launched. To execute the ***OntologyManagerTab*****,** the user must then open any project as in the sequence shown in **Figures 5** and **6**.

The user must then enable the ***OntologyManagerTab*** on the current project. To do so, he must follow the path "Project ➔ Configure", select the ***OntologyManagerTab*** to run as a Tab Widget and click "OK". This sequence is shown in **Figures 7**, **8** and **9**.

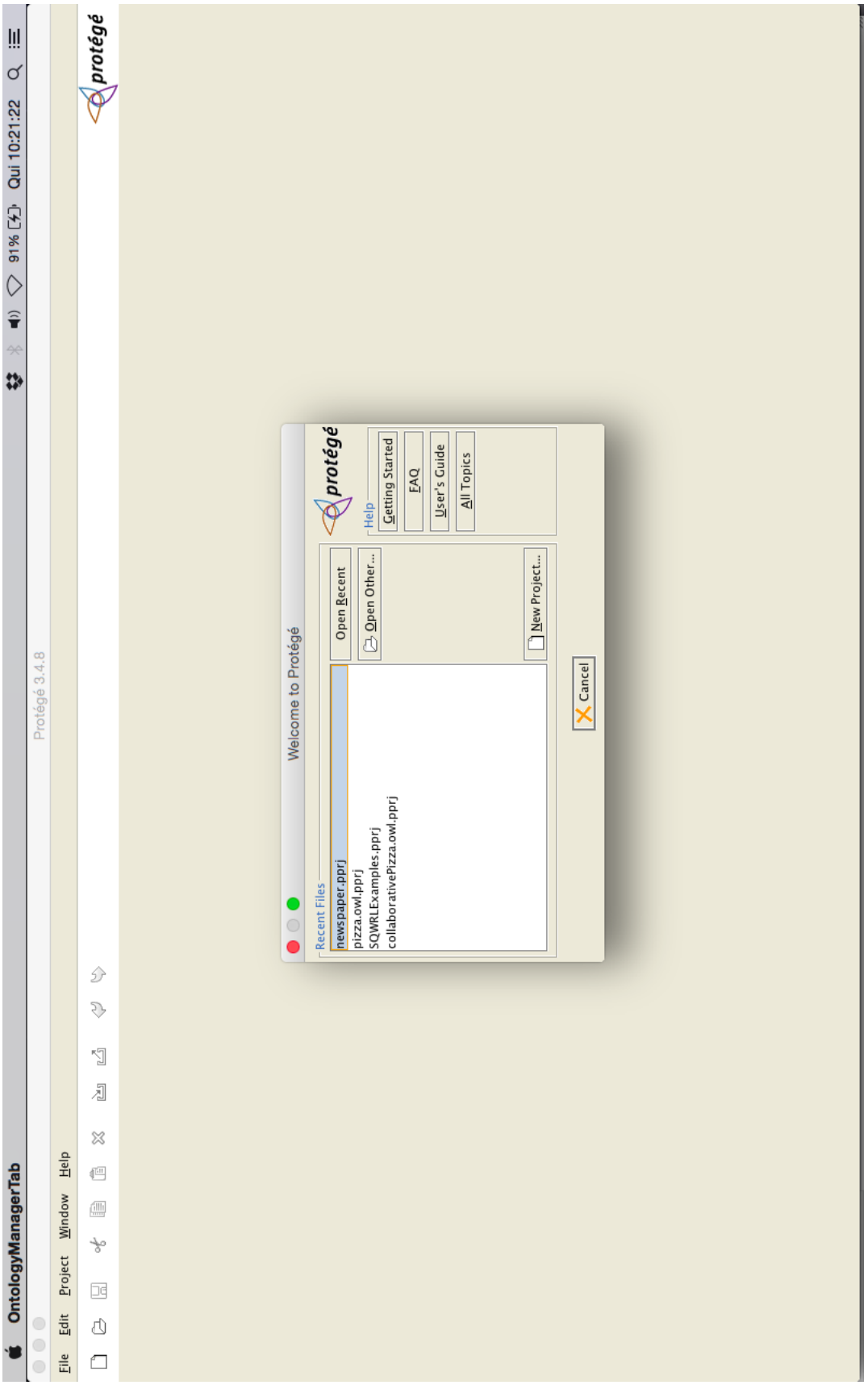

Figure 5. Launching Protégé

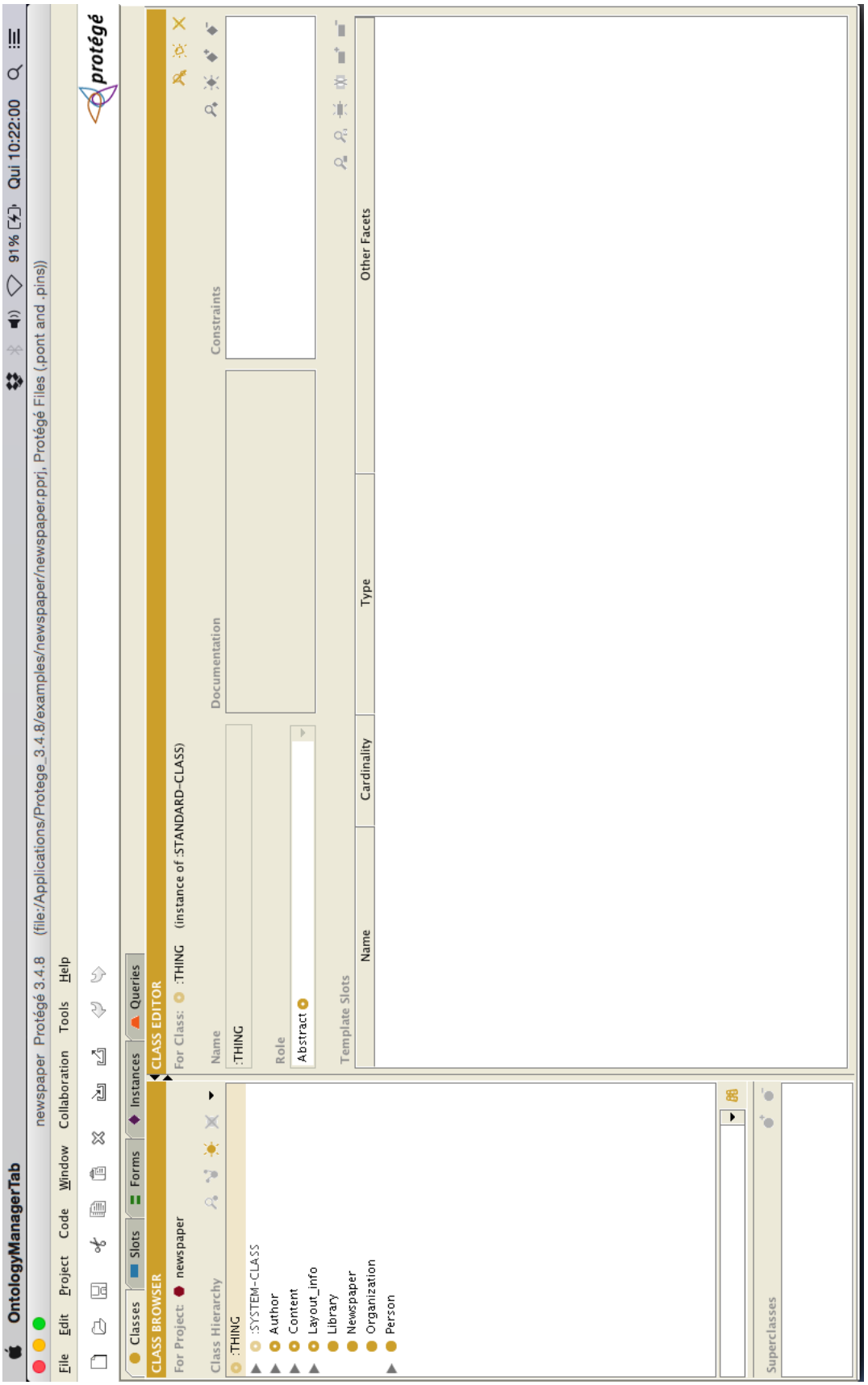

Figure 6. Opening Protégé Project

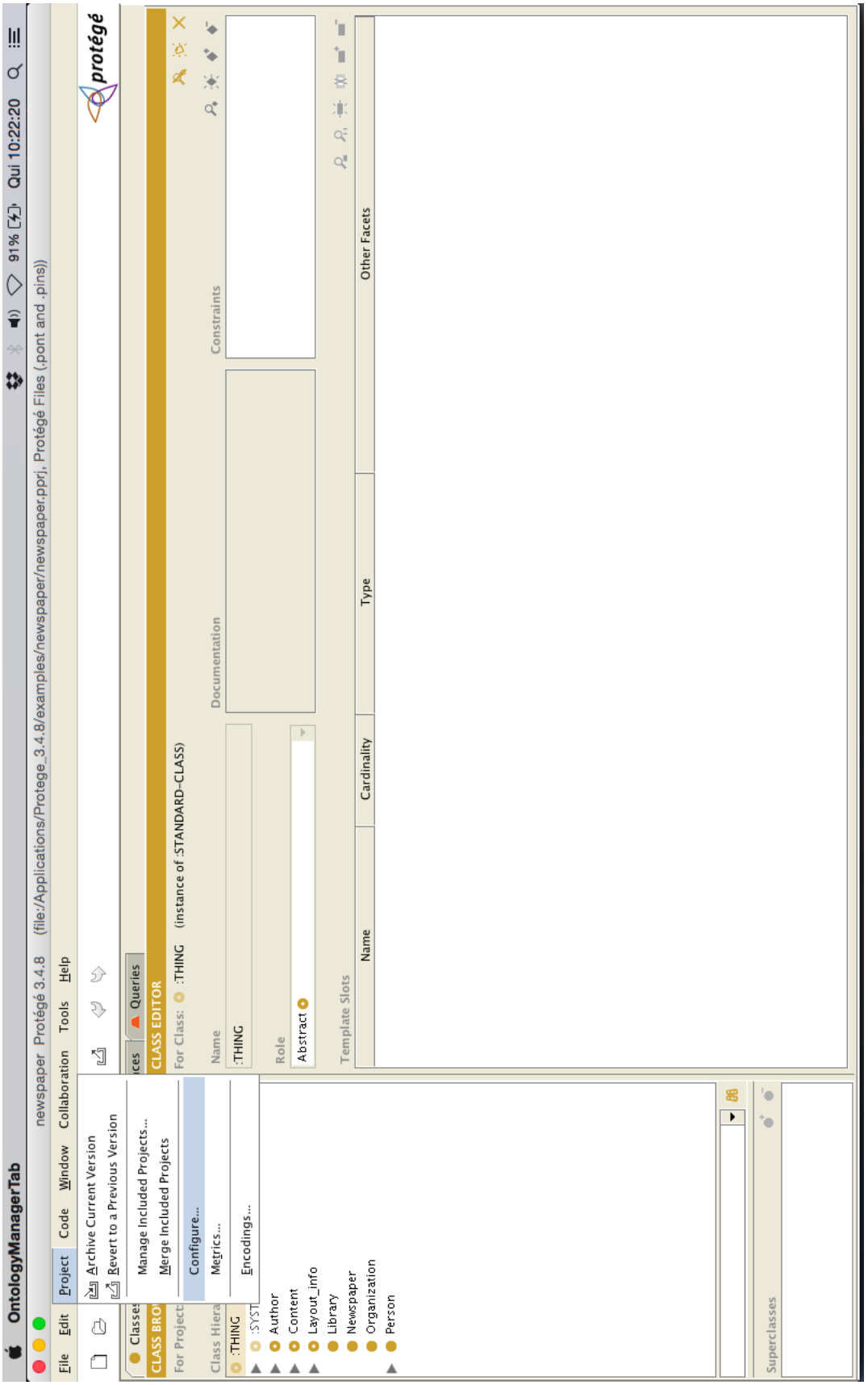

Figure 7. Configuring Protégé Widgets

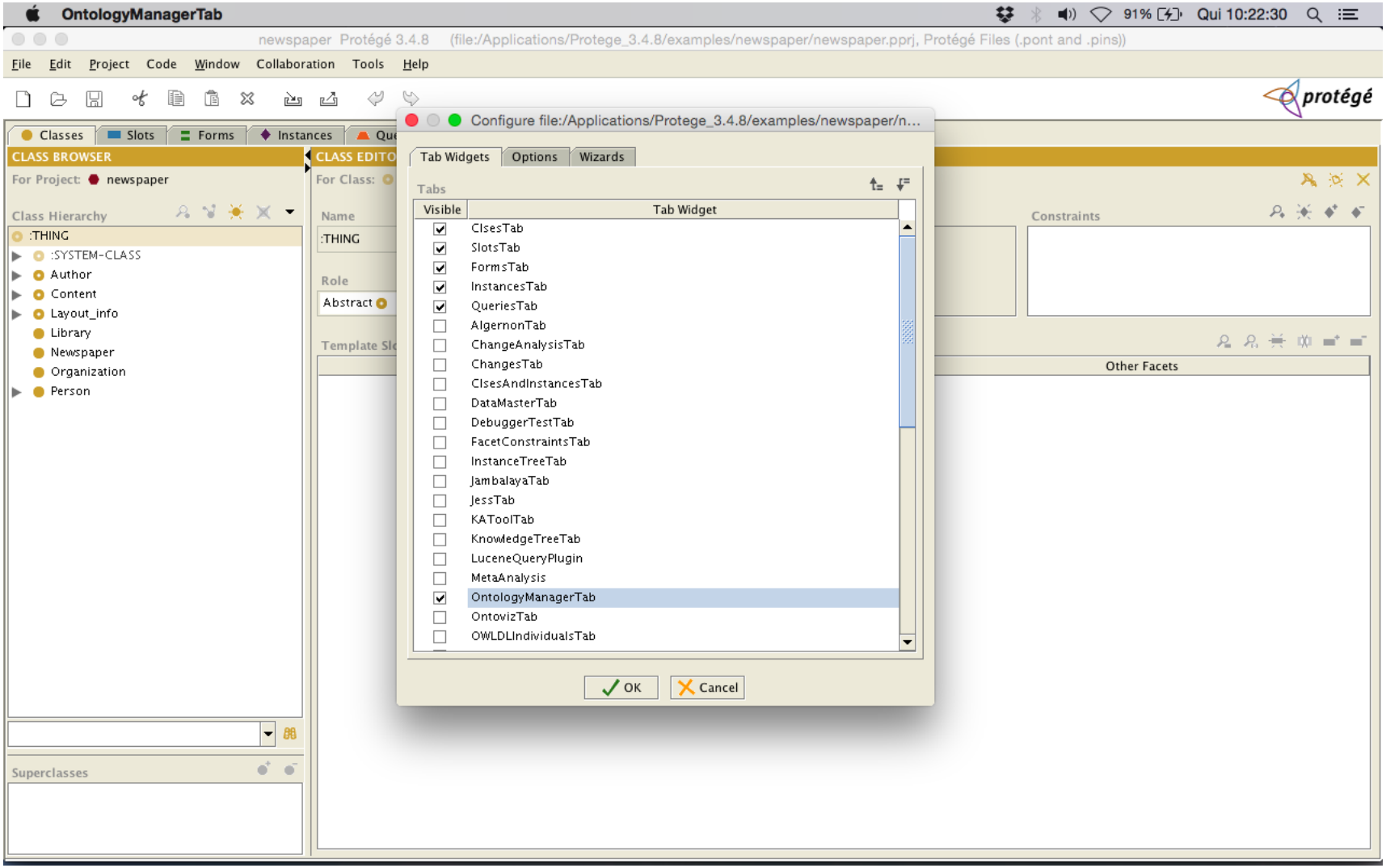

Figure 8. Selecting ***OntologyManagerTab***

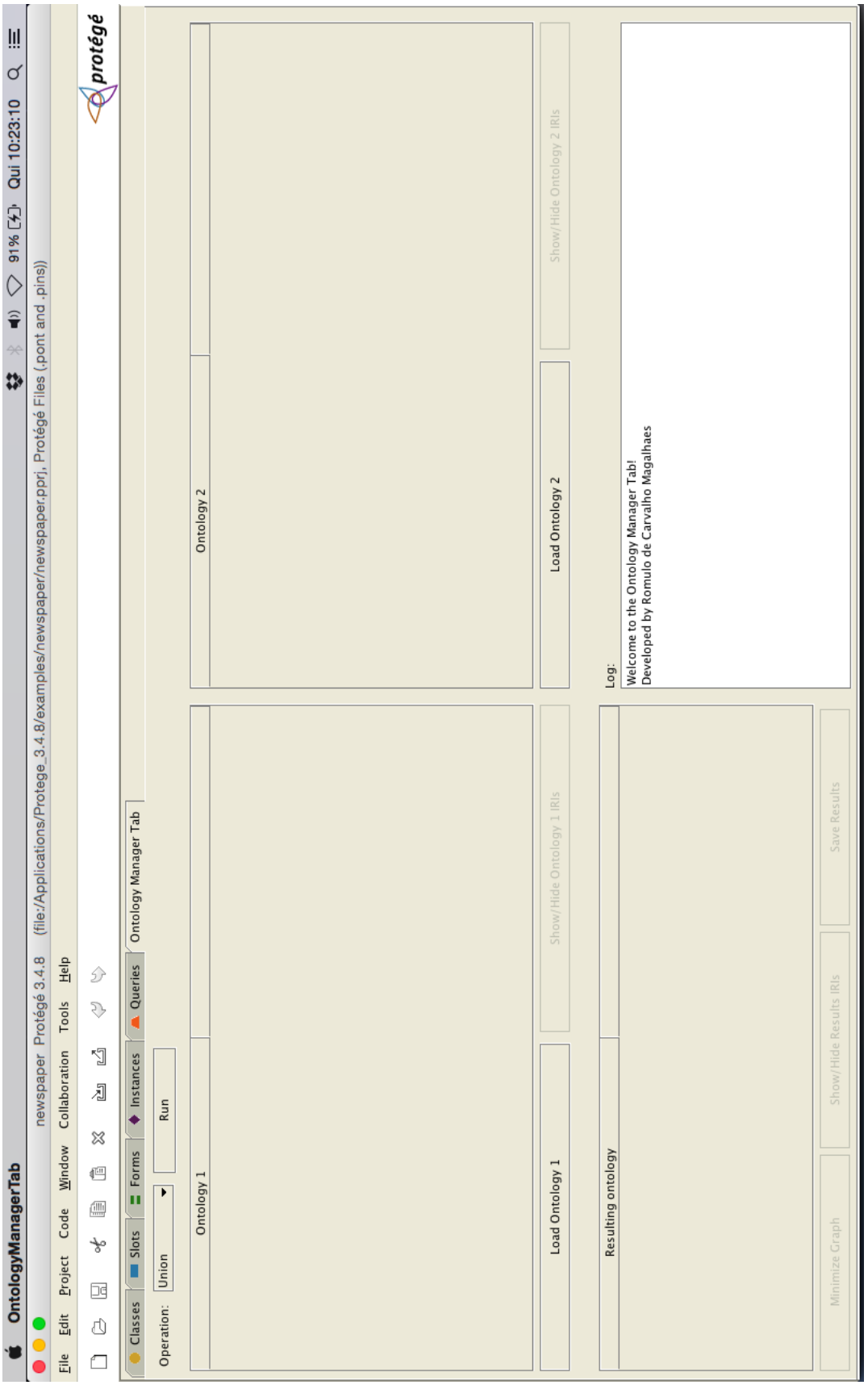

Figure 9. ***OntologyManagerTab***

6.4 Software Usage

In this section, we will explain how the software works using examples. This section covers the basic functionalities of ***OntologyManagerTab***: loading ontologies; the Union procedure; the Intersection procedure; the Projection procedure; the Difference procedure; minimizing ontologies; and finally saving the resulting ontology.

Furthermore, we must keep in mind that in contrast to what is presented in the examples in Section 5 the ***OntologyManagerTab*** will also show the "mirror" of the ontologies as specified in **Definition 1.**

6.4.1. Loading Ontologies

To load an OWL ontology, the user must simply click on one of the loading buttons – "Load Ontology 1", "Load Ontology 2" or "Load Ontology Projection" (in the case of the **Projection** operation) – and select the target ontology. The software will normalize this ontology, according to the rules in **Table 1** of Section 2.5, save this normalized lightweight core of the ontology in a file, with the same name as the original, minus the extension, concatenated with "Normalized.owl", and load the normalization result.

As an example, we will load a version of the FOAF ontology as *Ontology 1*, as shown in **Figures 10** and **11**. The representation adopted to display an ontology to the user utilizes a table format (see **Figure 11)**, where:

- Each lightweight term used in a constraint of the ontology appears at least once in the first column of the table; and
- Each constraint of the form $\boldsymbol{e} \sqsubseteq \boldsymbol{f}$ appears in a separate row, where $\boldsymbol{e}$ appears in column 1 and $\boldsymbol{f}$ in column 2.

By default, each term is displayed with their full URI. However, to facilitate their visualization, each ontology table has a "Show/Hide Ontology IRIs" button, whose functionality is exemplified in **Figures 11** and **12**.

After one ontology is loaded we can also see that the normalized file for said ontology is created as specified earlier. This can be seen for the FOAF ontology loaded in **Figure 13**.

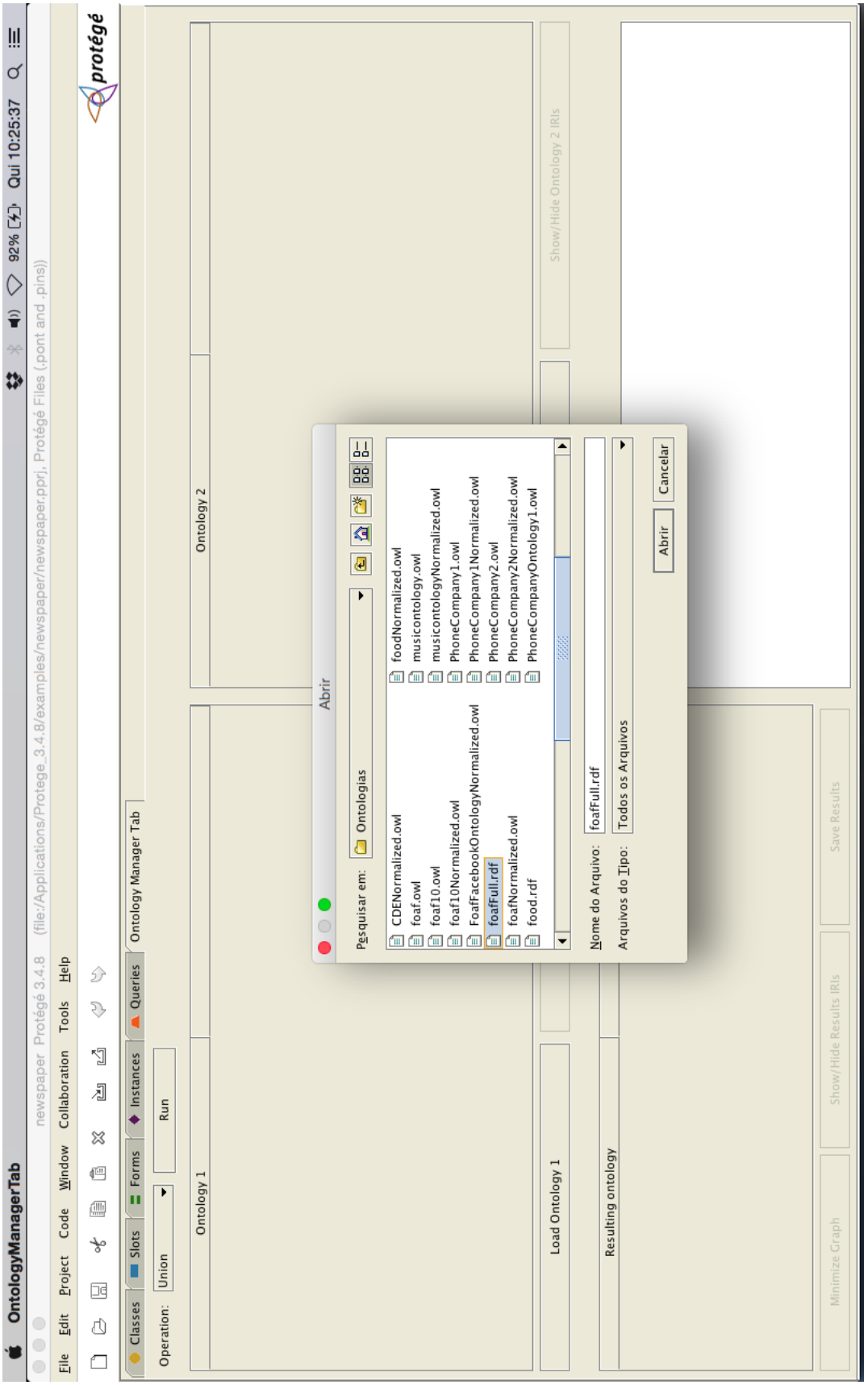

Figure 10. Loading version of FOAF Ontology

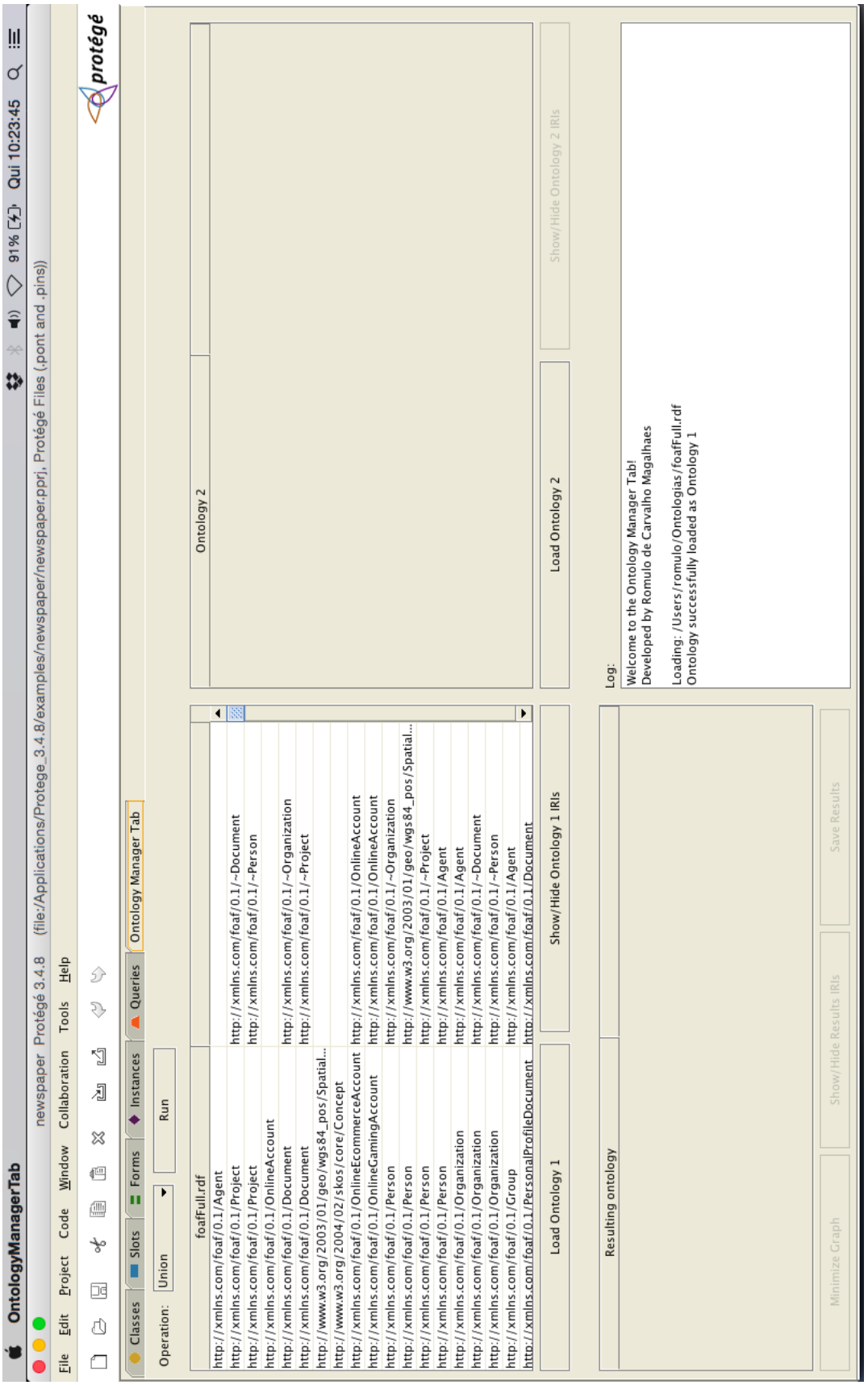

Figure 11. Version of FOAF Loaded

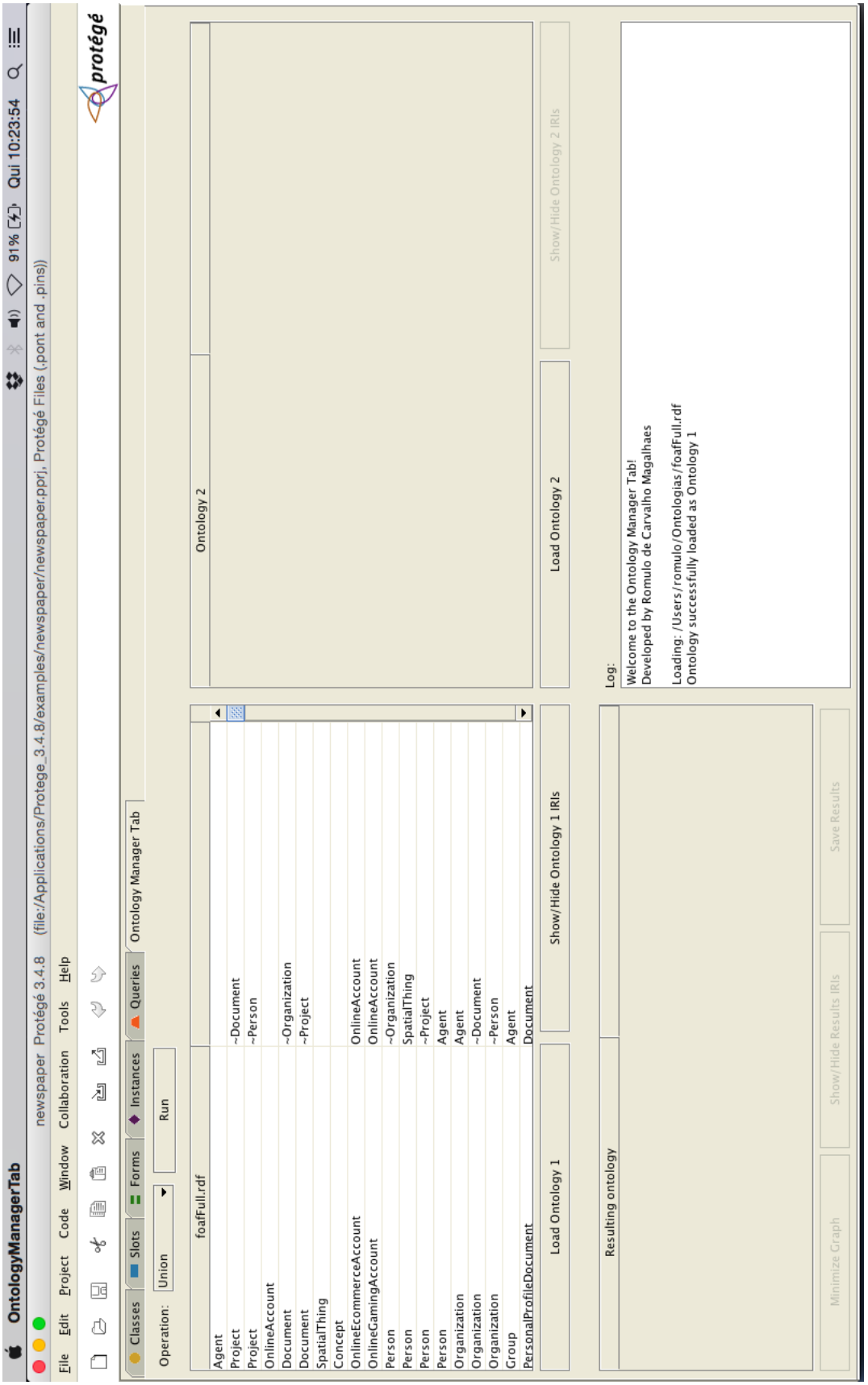

Figure 12. Hiding FOAF IRIs

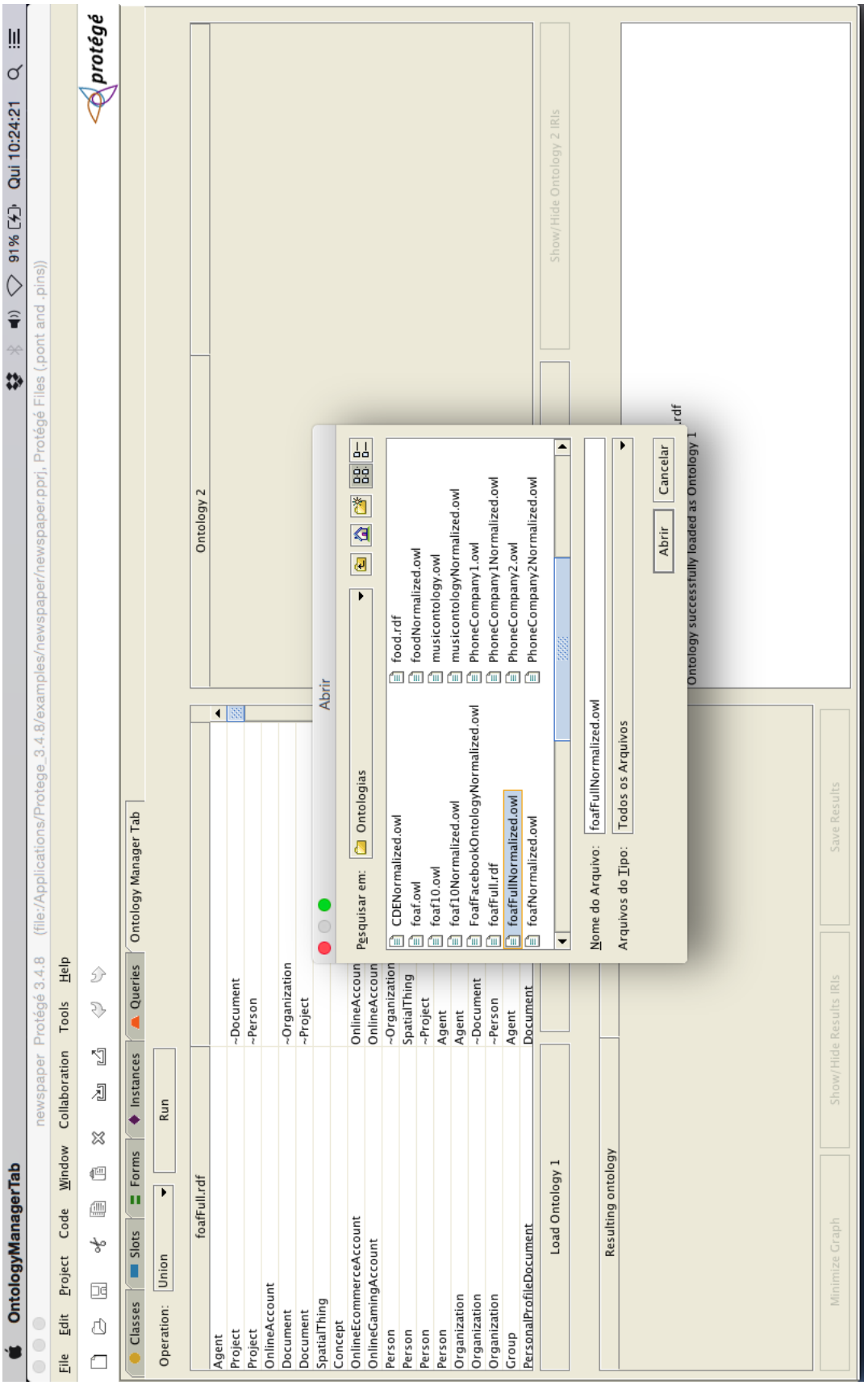

Figure 13. FOAF normalized file created

6.4.2. The Union Procedure

The **Union** procedure uses the algorithm specified in Sub-Section 5.4 and is the default operation for the ***OntologyManagerTab***. The **Union** is the first option in the software operation combo box. To exemplify its usage, we will simulate the **Union** from **Example 2** that appears in Section 5.4 and uses two fragments of the FOAF ontology. All figures in this example will be shown without full URIs to facilitate the visualization.

First, we load O_1 as Ontology 1, as shown in **Figure 14**. Next, we load O_2 as Ontology 2, as shown in **Figure 15**. Finally, we click the Run button to execute the **Union** procedure. The resulting ontology is shown in **Figure 16**, as seen in **Example 2**. If the user tries to execute the **Union** operation without loading two ontologies, the ***OntologyManagerTab*** will show a warning asking for the second ontology, as shown in **Figure 17**.

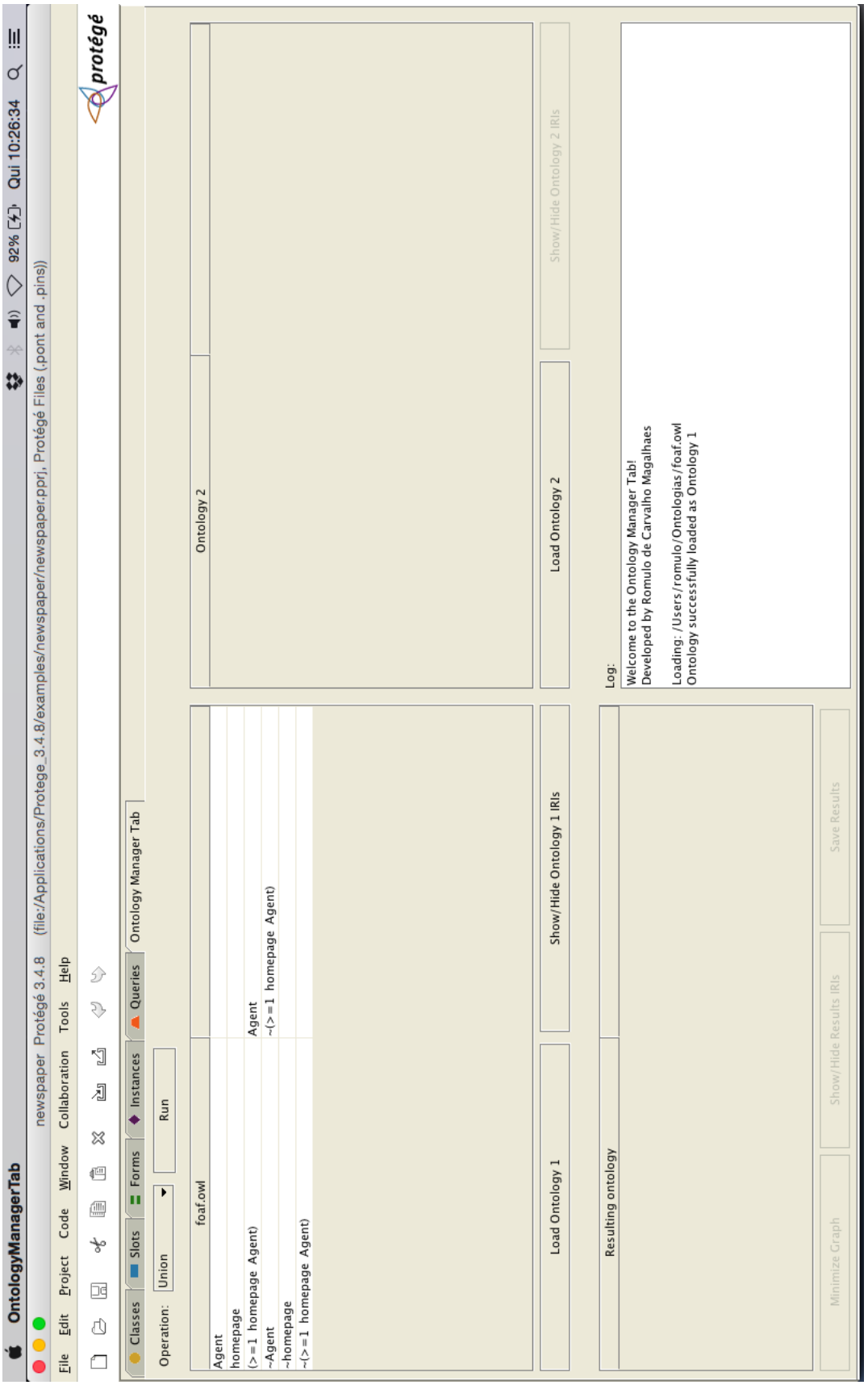

Figure 14. O_1 loaded as Ontology 1

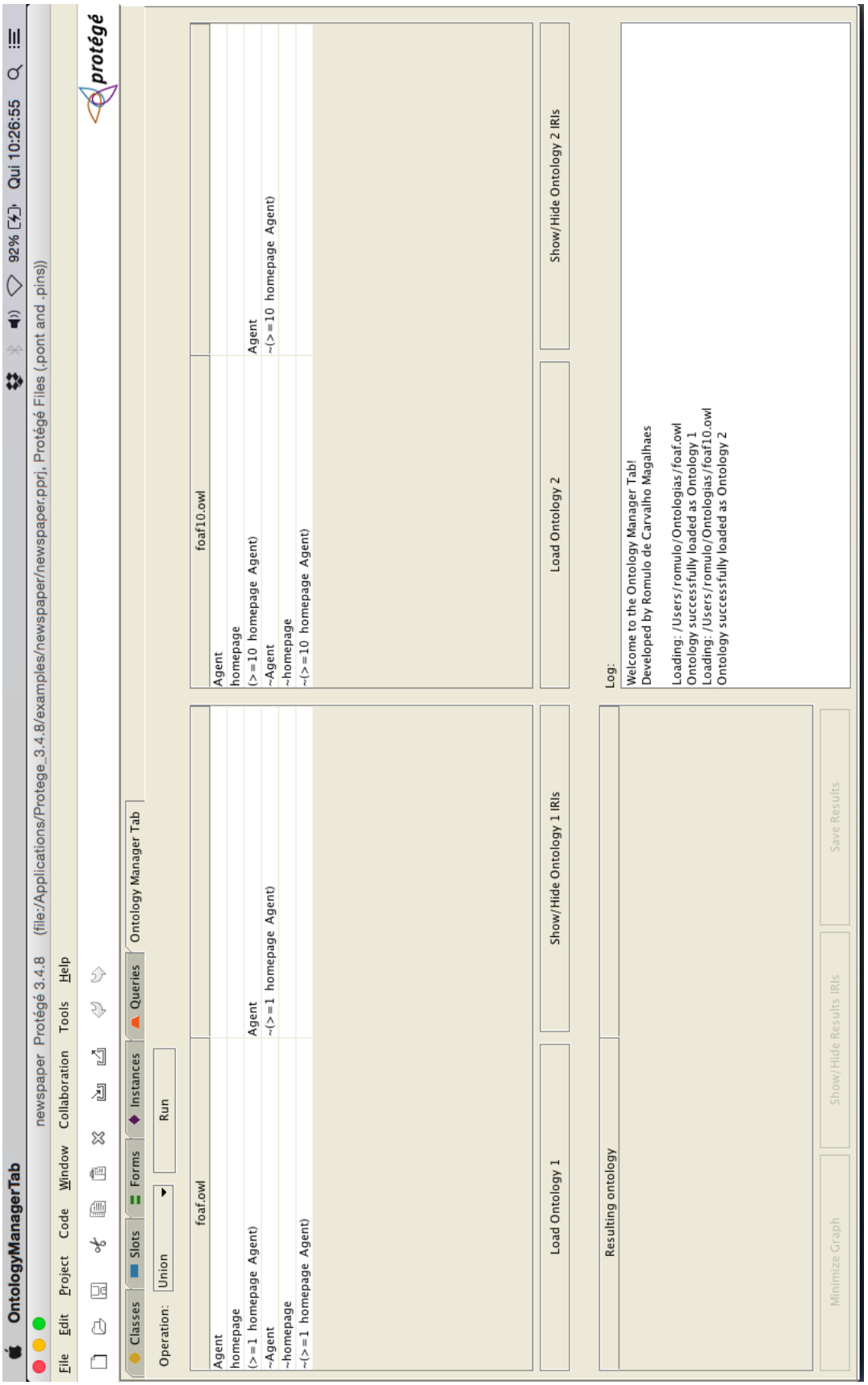

Figure 15. O_2 loaded as Ontology 2

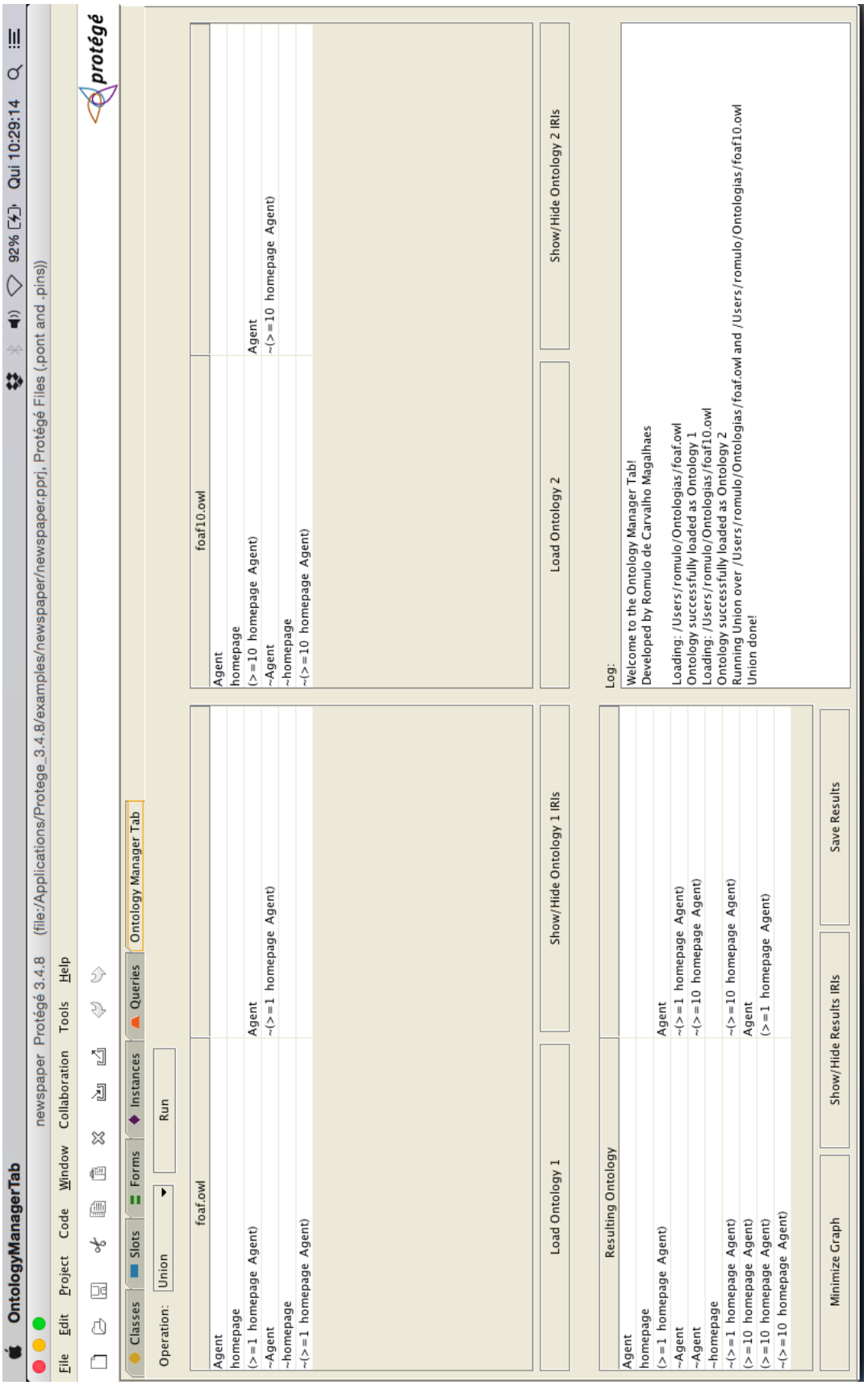

Figure 16. Resulting Ontology for the **Union** of O_1 and O_2

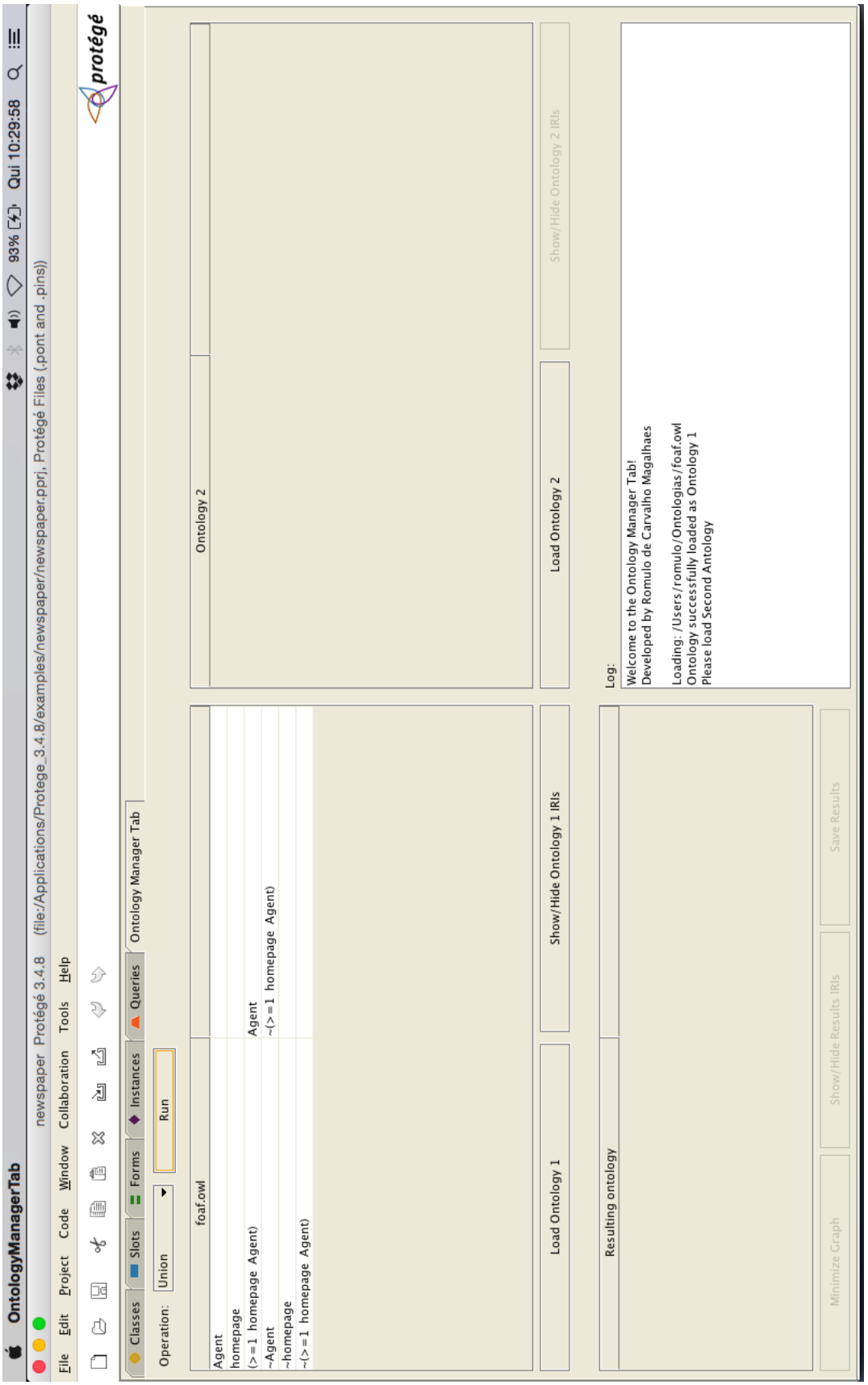

Figure 17. ***OntologyManagerTab*** asking for second ontology

6.4.3. The Intersection procedure

The **Intersection** procedure uses the algorithm specified in Section 5.5 and is the second option in the software operation combo box. To exemplify its usage, we will simulate the **Intersection** from **Example 5** that appears in Section 5.5 and uses two ontologies based on phone companies, $O_{PhoneCompany1}$ and $O_{PhoneCompany2}$. Again all figures in this example will be shown without full URIs to facilitate the visualization.

First, we load $O_{PhoneCompany1}$ as Ontology 1, as shown in **Figure 18**. Next, we load $O_{PhoneCompany2}$ as Ontology 2, as shown in **Figure 19**. Then, we select the **Intersection** operation in the combo box. Finally, we click the Run button to execute the **Intersection** procedure. The resulting ontology is shown in **Figure 20**, as seen in **Example 5**. As in the **Union** procedure, if the user tries to execute the **Intersection** operation without loading two ontologies, the ***OntologyManagerTab*** will show a warning asking for the second ontology, as shown in **Figure 17.**

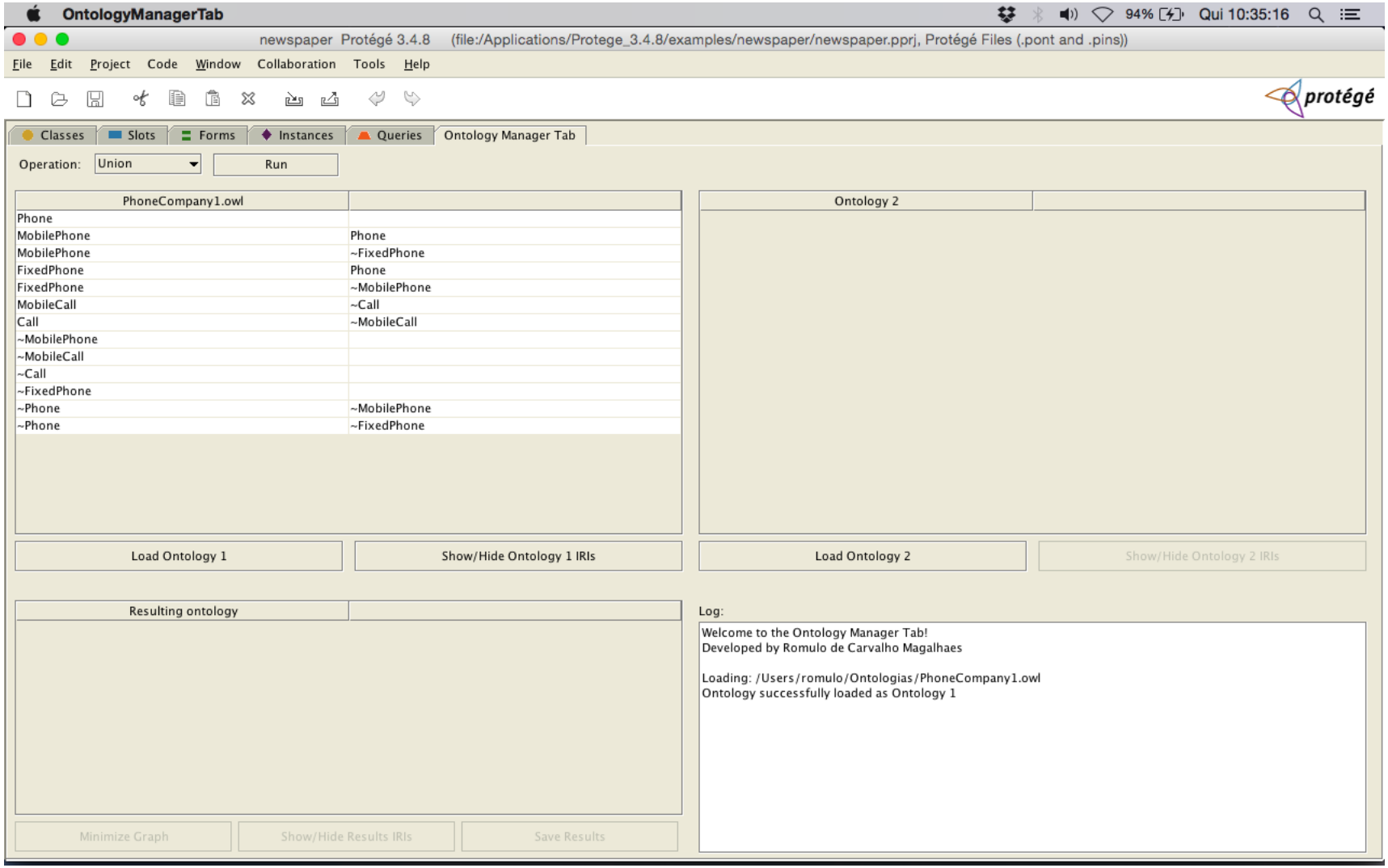

Figure 18. *O PhoneCompany1* loaded as Ontology 1

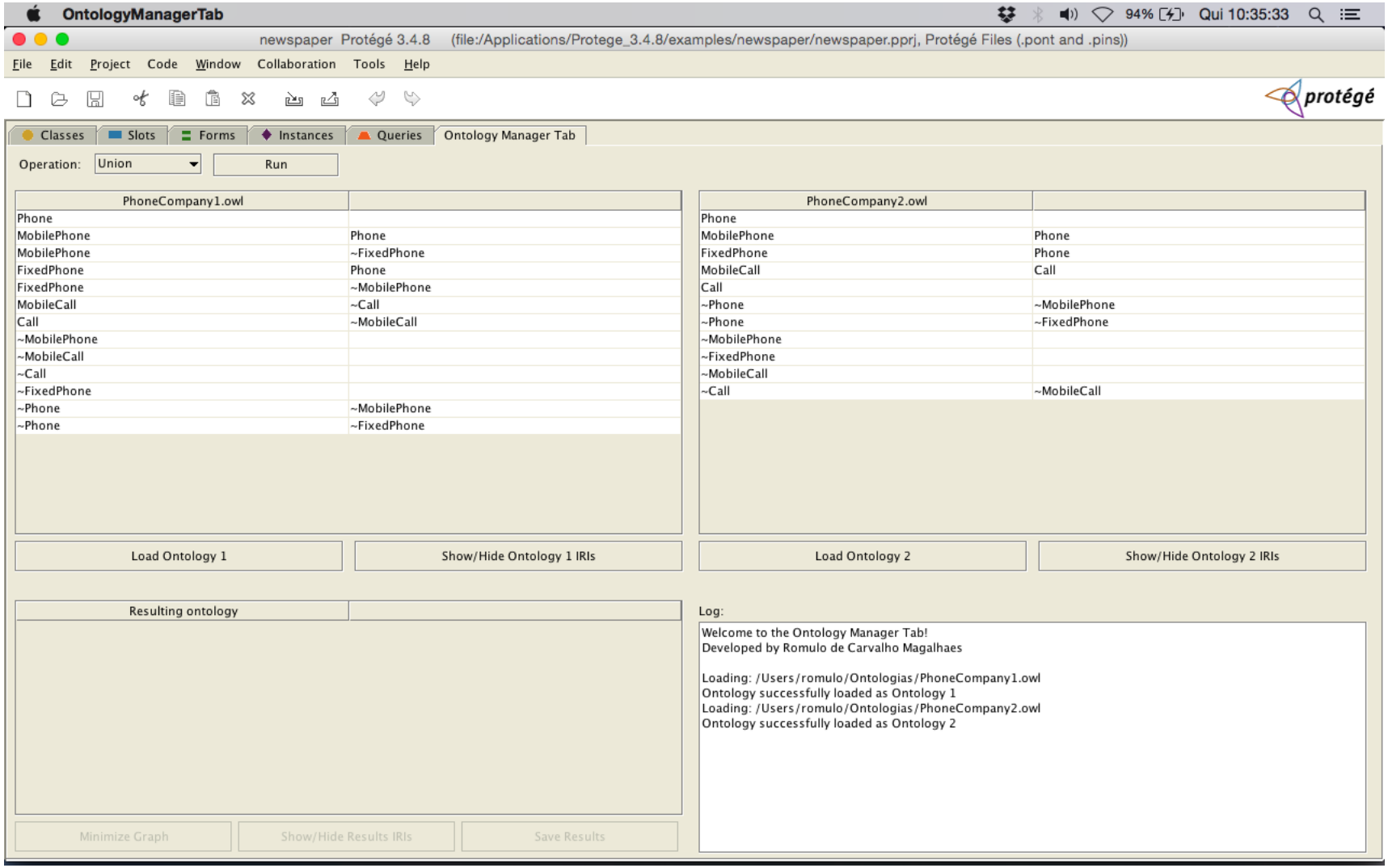

Figure 19. *O*$_{PhoneCompany2}$ loaded as Ontology 2

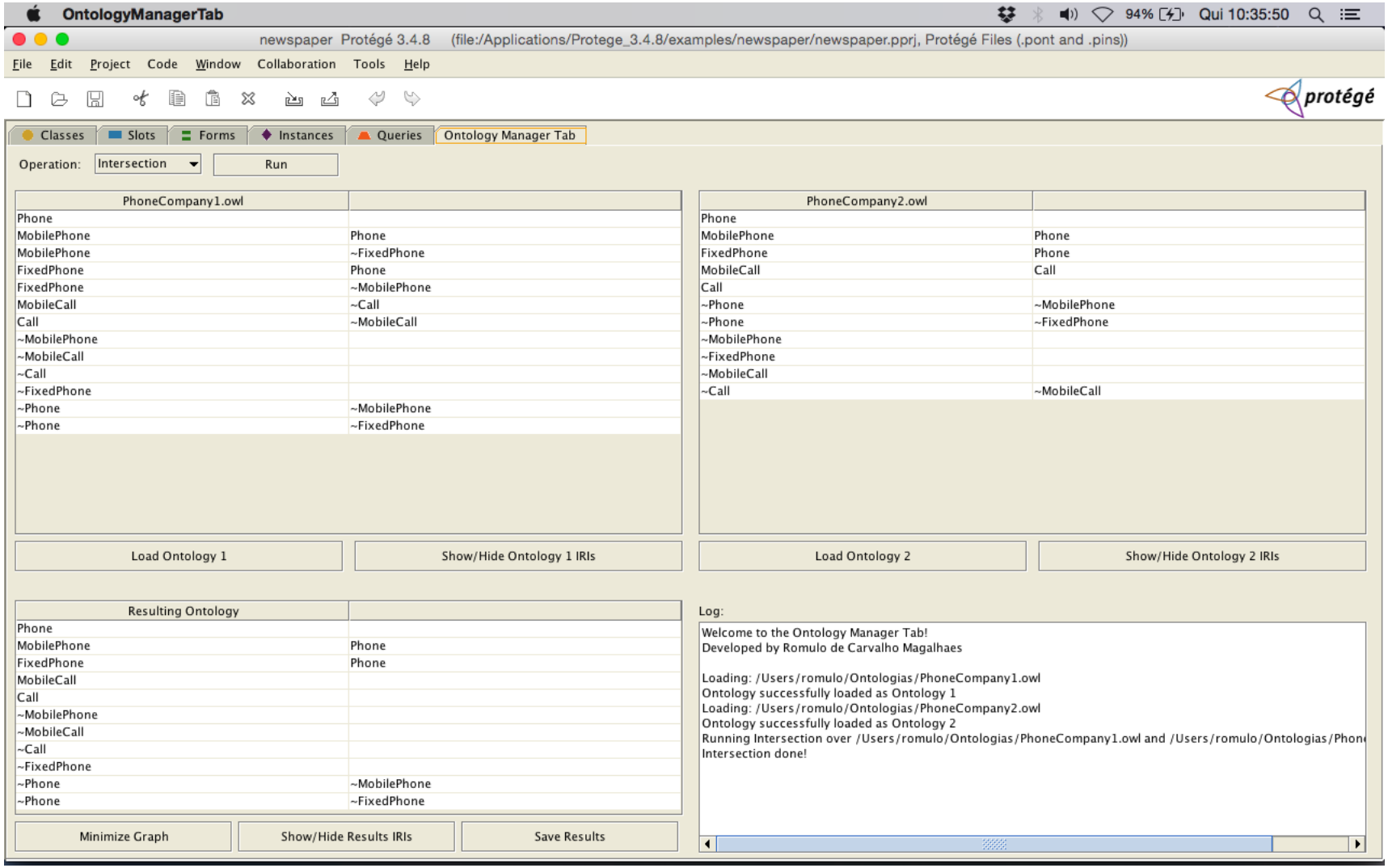

Figure 20. Resulting Ontology for the **Intersection** of $O_{PhoneCompany1}$ and $O_{PhoneCompany2}$

6.4.4. The Projection procedure

The **Projection** procedure uses the algorithm specified in Section 5.3 and is the third option in the software operation combo box. To exemplify its usage, we will simulate the **Projection** from **Example 1** that appears in Section 5.3 and executes the projection of the FOAF ontology over V_{FF} = {foaf:Agent, foaf:Person, foaf:Organization, foaf:account }. Again all figures in this example will be shown without full URIs to facilitate the visualization.

First, we load the FOAF ontology as Ontology 1, as shown in **Figure 21**. Next, we select the **Projection** operation in the combo box. Then, we select the terms from V_{FF} that will be used in the projection, as shown in **Figure 22**. Finally, we click the Run button to execute the **Projection** procedure. The resulting ontology is shown in **Figure 23**, as seen in **Example 1**.

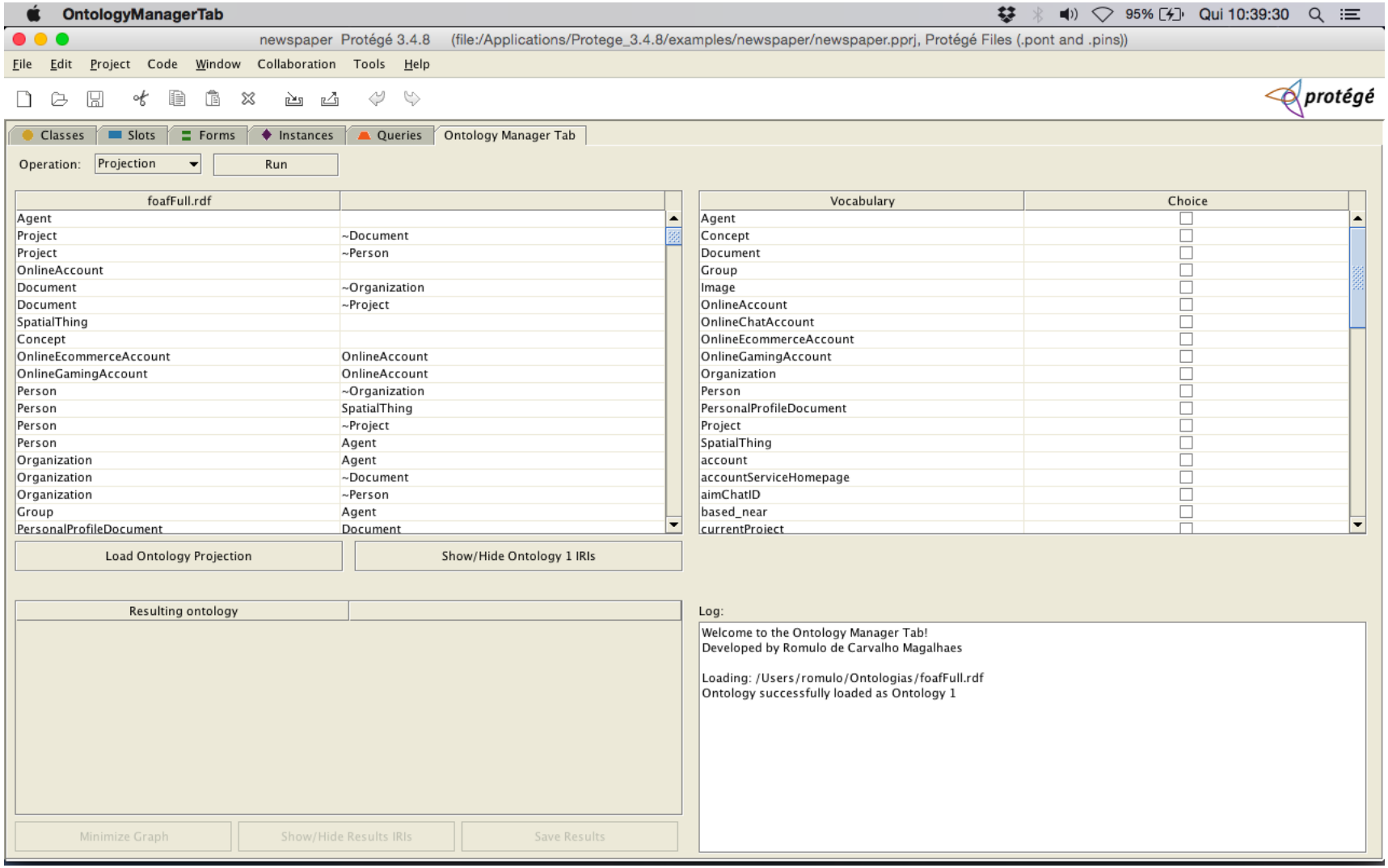

Figure 21. FOAF Ontology loaded as Ontology 1

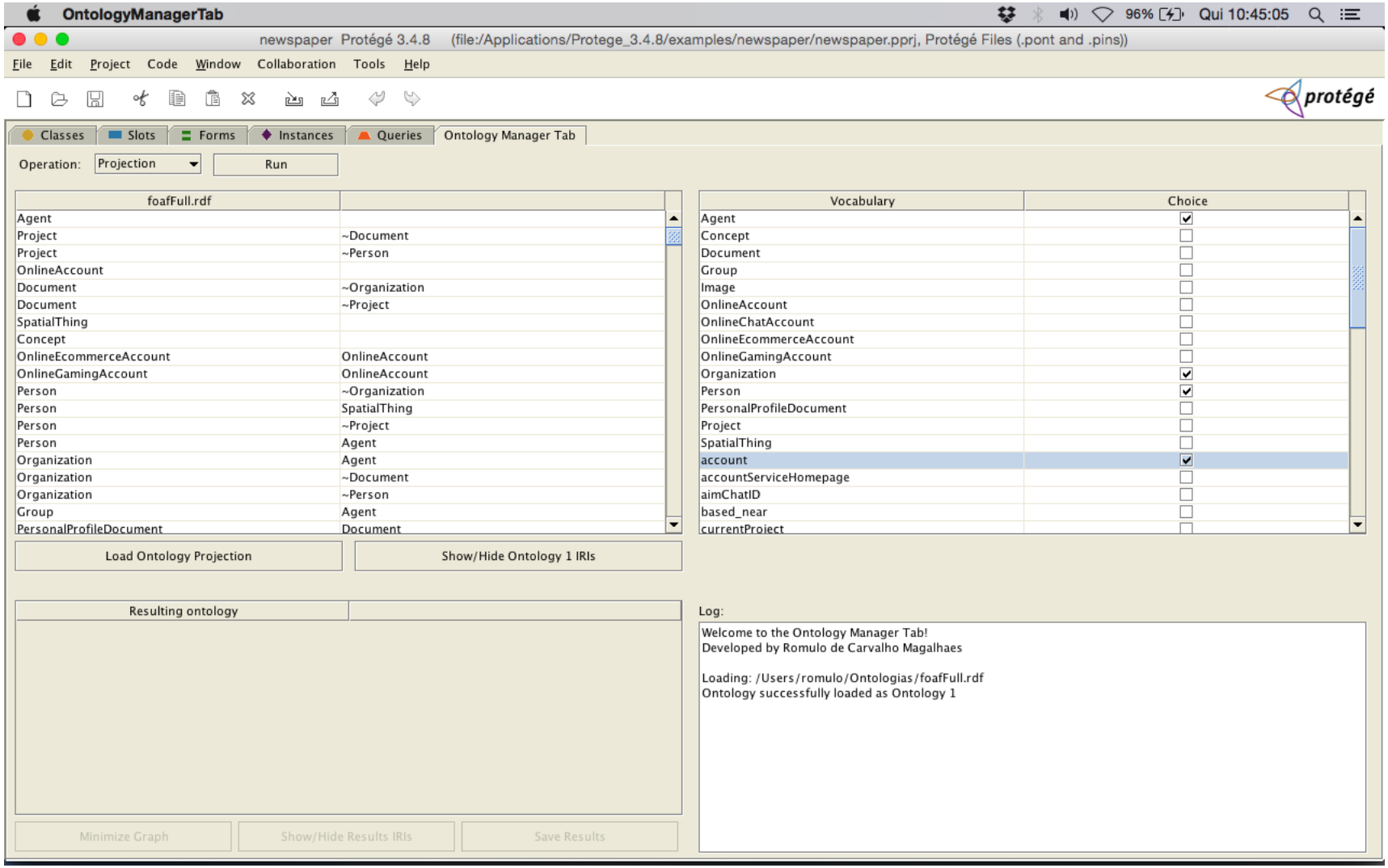

Figure 22. Selection of V_{FF} for the **Projection** procedure

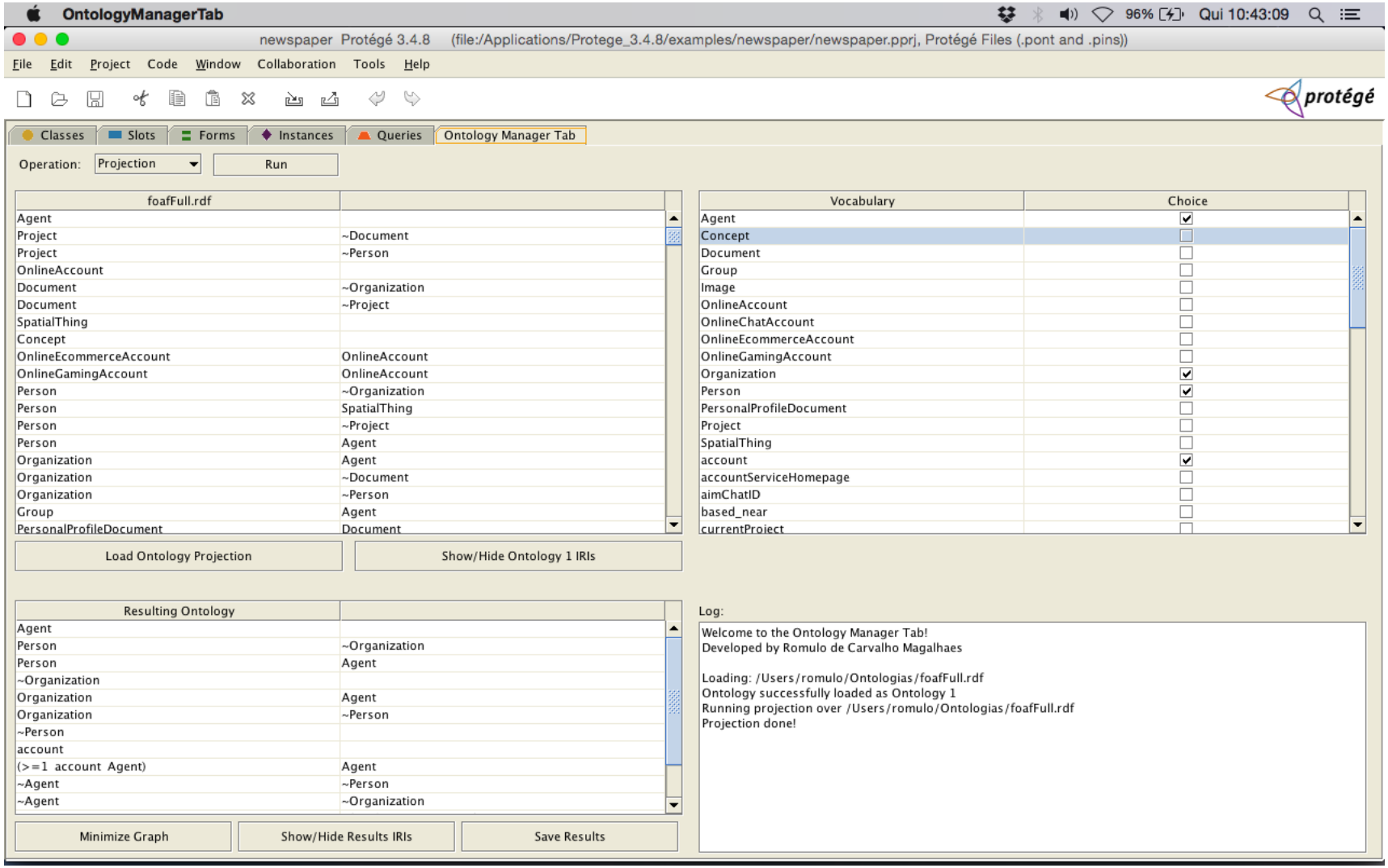

Figure 23. Resulting Ontology for the **Projection** of V_{FF} over the FOAF ontology

6.4.5. The Difference procedure

The **Difference** procedure uses the algorithm specified in Section 5.6 and it is the fourth and last option in the software operation combo box. To exemplify its usage, we will simulate the **Difference** from **Example 7** that appears in Section 5.6 and uses two ontologies based on phone companies, $O_{PhoneCompany1}$ and $O_{PhoneCompany2}$. Again all figures in this example will be shown without full URIs to facilitate the visualization.

First, we load $O_{PhoneCompany1}$ as Ontology 1, as shown in **Figure 18**. Next, we load $O_{PhoneCompany2}$ as Ontology 2, as shown in **Figure 19**. Then, we select the **difference** operation in the combo box. Finally, we click the Run button to execute the **Difference** procedure. The resulting ontology is shown in **Figure 24**, as seen in **Example 7**. As in the **Union** procedure, if the user tries to execute the **Intersection** operation without loading two ontologies, the ***OntologyManagerTab*** will show a warning asking for the second ontology, as shown in **Figure 17.**

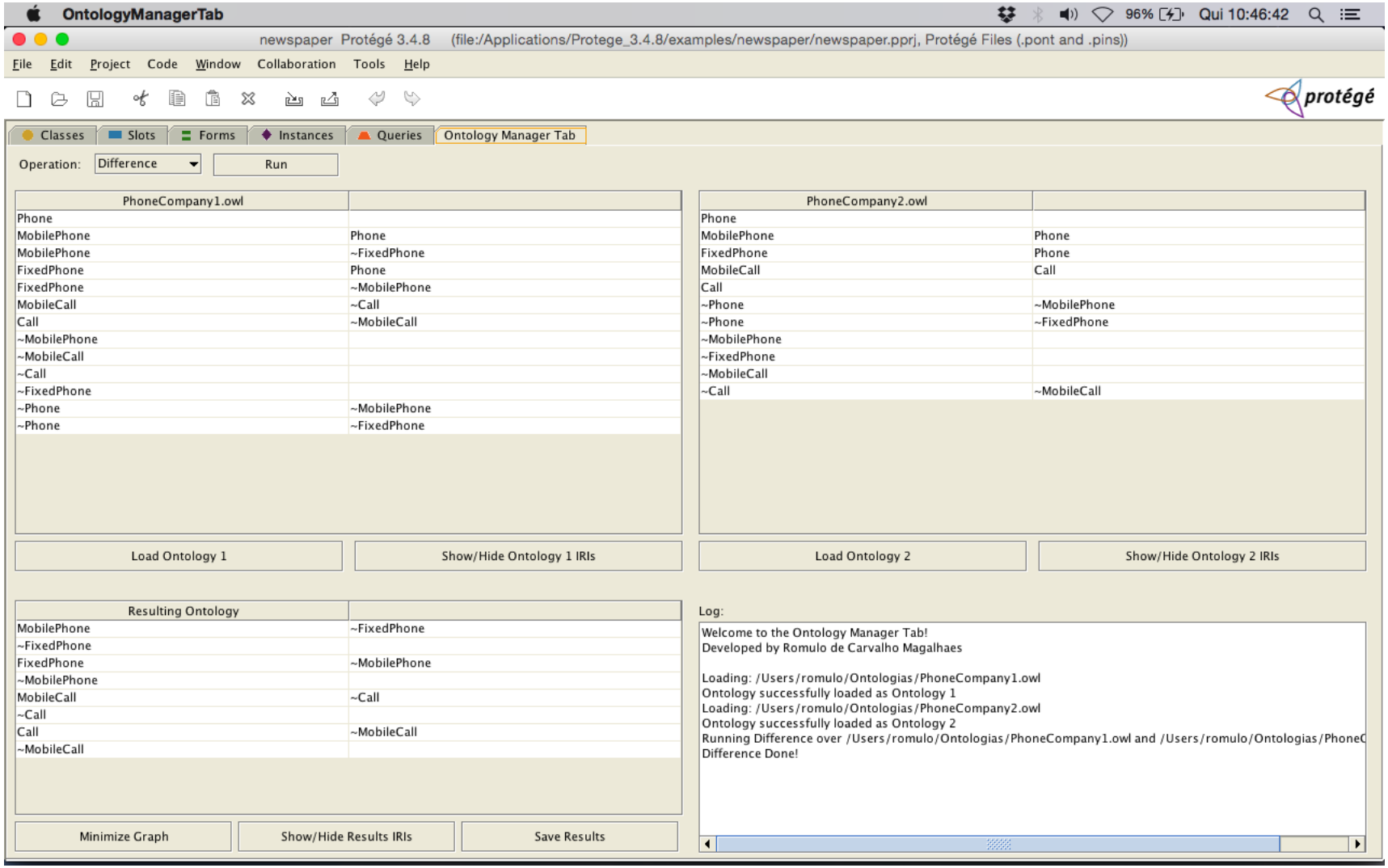

Figure 24. Resulting Ontology for the **Difference** of $O_{PhoneCompany1}$ and $O_{PhoneCompany2}$

6.4.6. Minimizing Ontologies

Our software provides a button to call the graph minimization function, as discussed in Section 4.3. This function is not automatically called, so that the user can follow the construction of the transitive closure of the resulting constraint graph when necessary.

To exemplify the use of the graph minimization function, we will consider the **Intersection** of two trivial ontologies O_1 and O_2, where O_1 represents A ⊑ B, B ⊑ C, C ⊑ D and O_2 represents A ⊑ B, B ⊑ C. **Figure 25** shows O_1 loaded as Ontology 1 and O_2 loaded as Ontology 2. **Figure 26** exhibits the result of the intersection, whose constraints are depicted in the bottom left table; note that this table contains a redundant constraint, A ⊑ C, which appears in Line 3. **Figure 27** shows the result of calling the graph minimization function; note that the bottom left table does not contain a line for the constraint A ⊑ C.

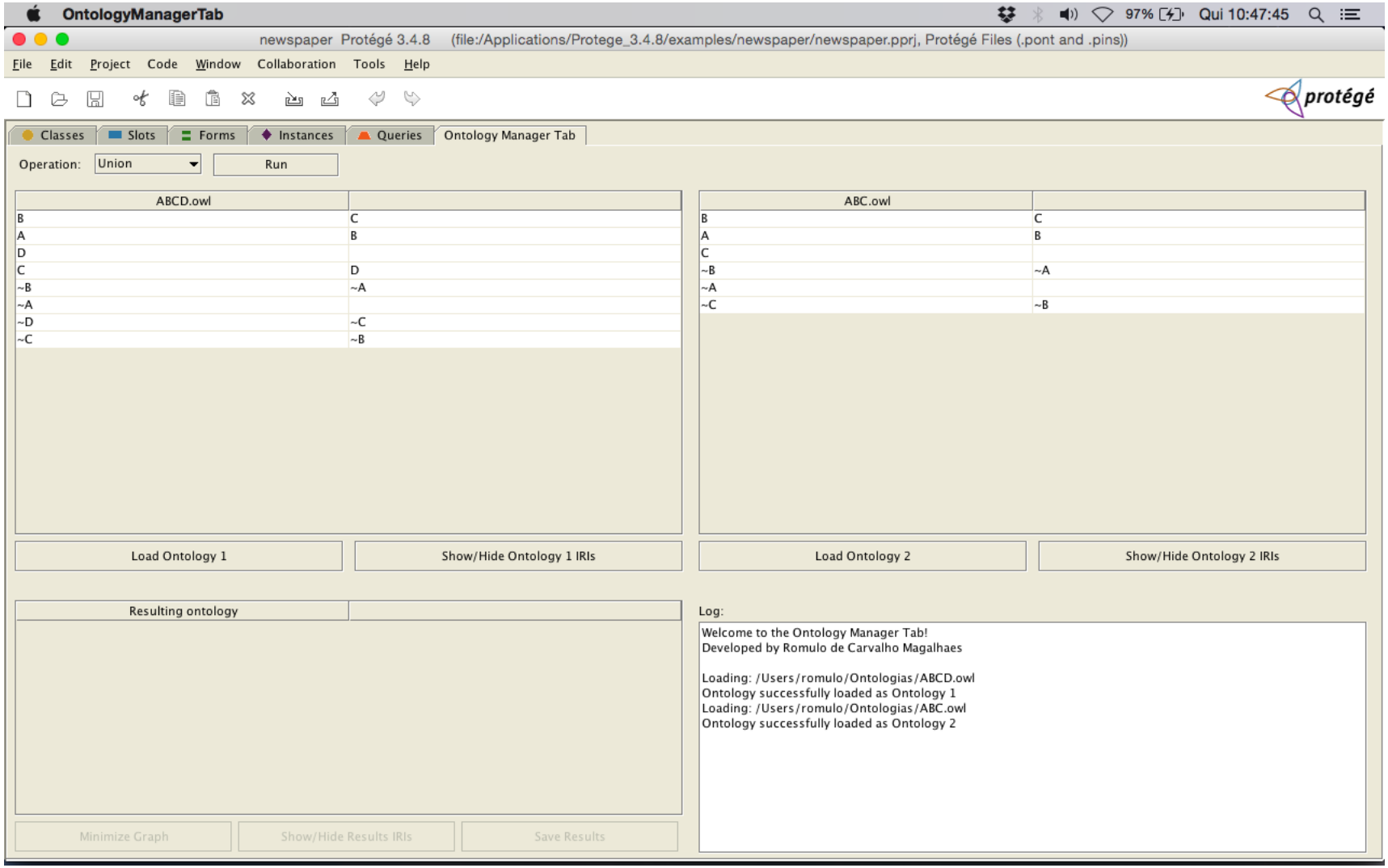

Figure 25. O_1 loaded as Ontology 1 and O_2 loaded as Ontology 2

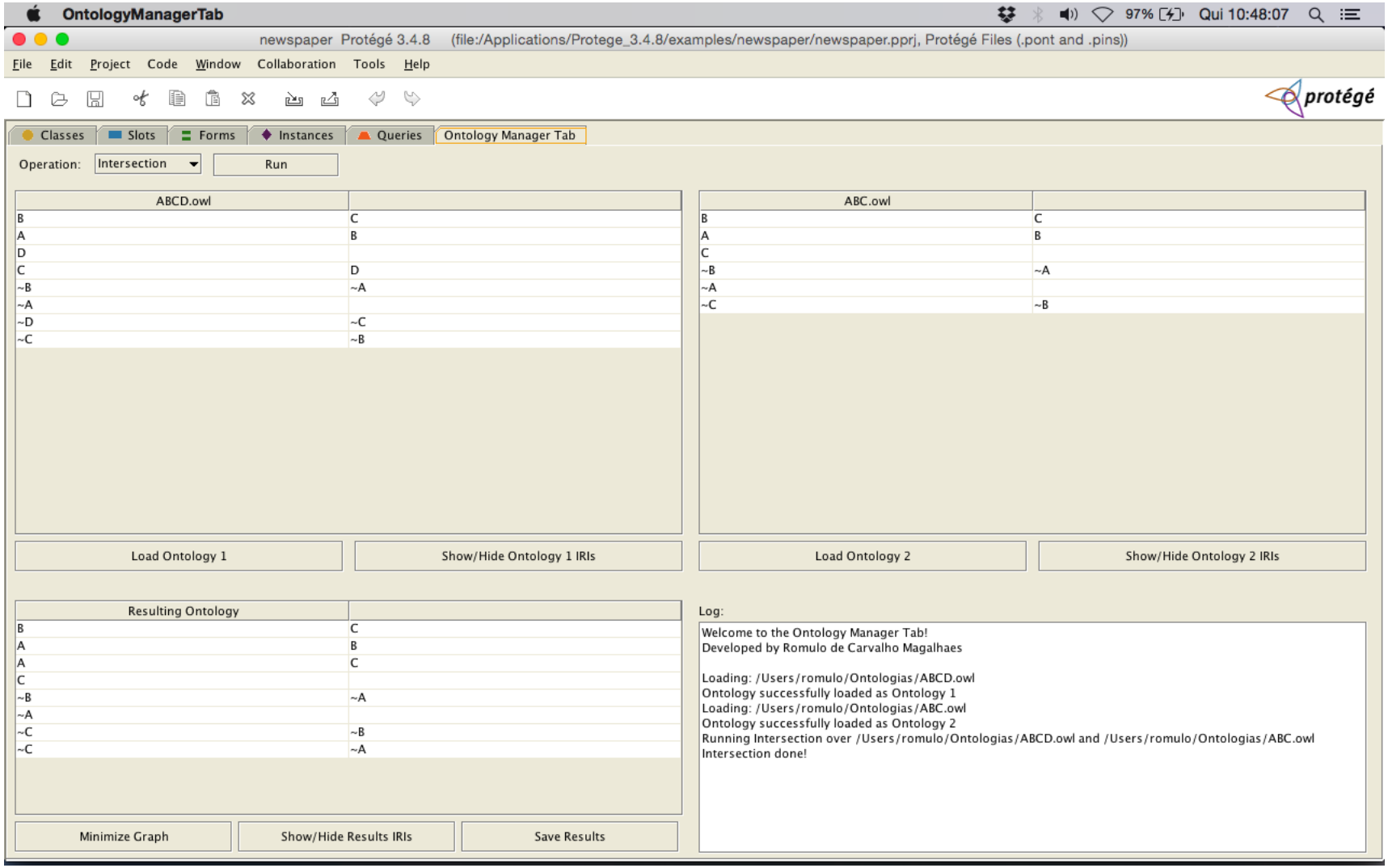

Figure 26. result of said **Intersection** between O_1 and O_2

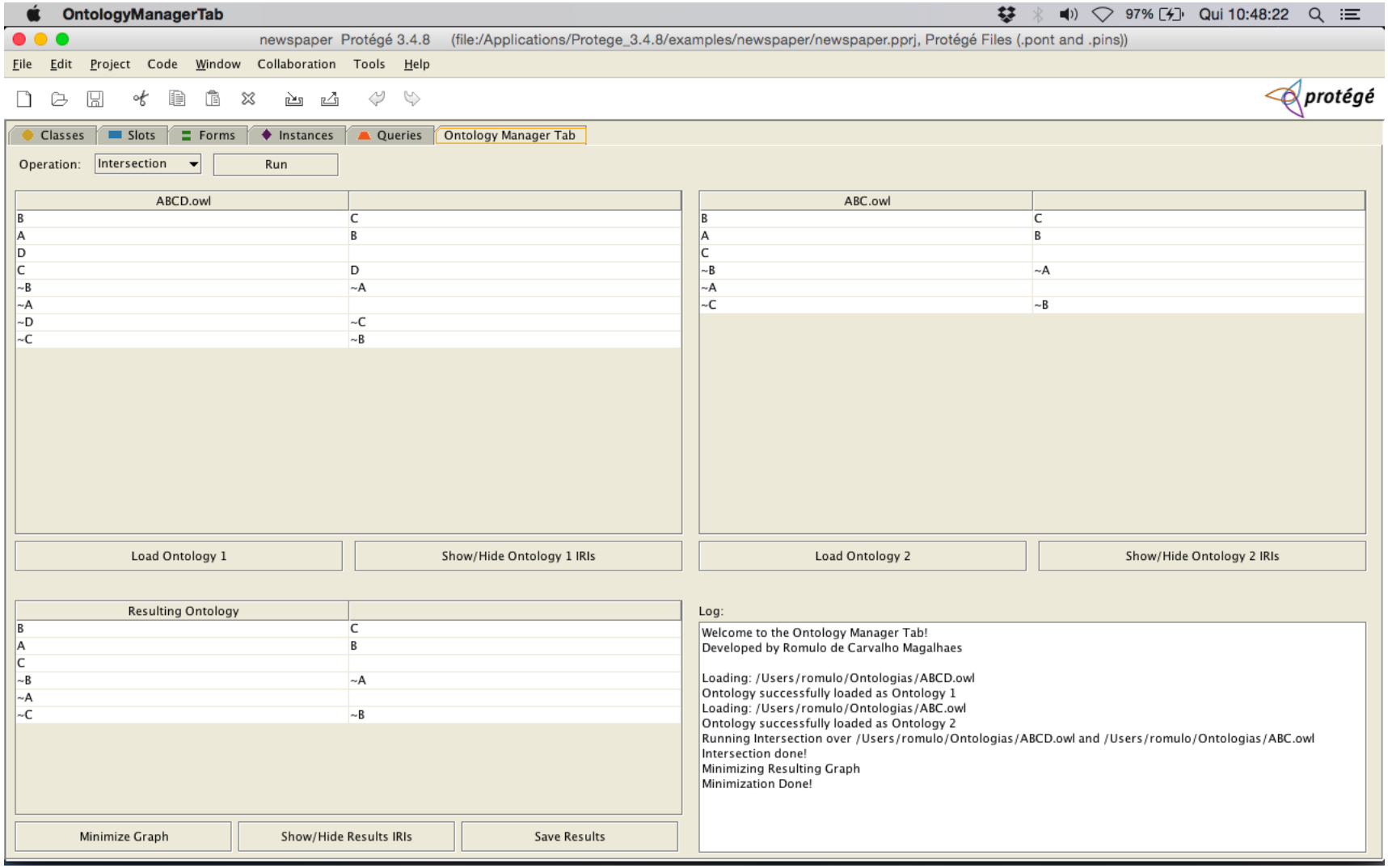

Figure 27. Minimize Graph function over **Intersection** between O_1 and O_2

The possibility of displaying all reachable nodes is very interesting when ⊥-nodes occur. ***OntologyManagerTab*** shows all ⊥-nodes found with their respective levels, as in **Definitions 2** and **3**. Every time the software finds ⊥-nodes, a warning is added to the log with the number of ⊥-nodes found in the last operation, including the load of an ontology whose constraints force a class to be empty (which is captured by ⊥-nodes).

If the graph minimization function was called after each operation, it would take the domain specialist a lot of unnecessary work to find the root of the ⊥-nodes. When the constraint graph has ⊥-nodes and is minimized, only the ⊥-node tags remain with their corresponding level.

Consider, for example, an **Union** between two ontologies O_3 and O_4, where O_3 contains the constraints A ⊑ B, B ⊑ C, B ⊑ ¬C and O_4 the constraints A ⊑ B, B ⊑ C, C ⊑ D, C ⊑ ¬D. **Figure 28** shows the result of their union and **Figure 29** the minimized graph. In this case, O_3 has two ⊥-nodes, where A is a ⊥-node of level 1 and B is a ⊥-node of level 0. O_4 has three ⊥-nodes, where A is a ⊥-node *of* level 2, B is a ⊥-node of level 1; and C is a ⊥-node of level 0. The resulting ontology will also have three ⊥-nodes, but in this case they will have different levels: A will be a ⊥-node of level 1, B will be a ⊥-node of level 0 and C will be a ⊥-node of level 0.

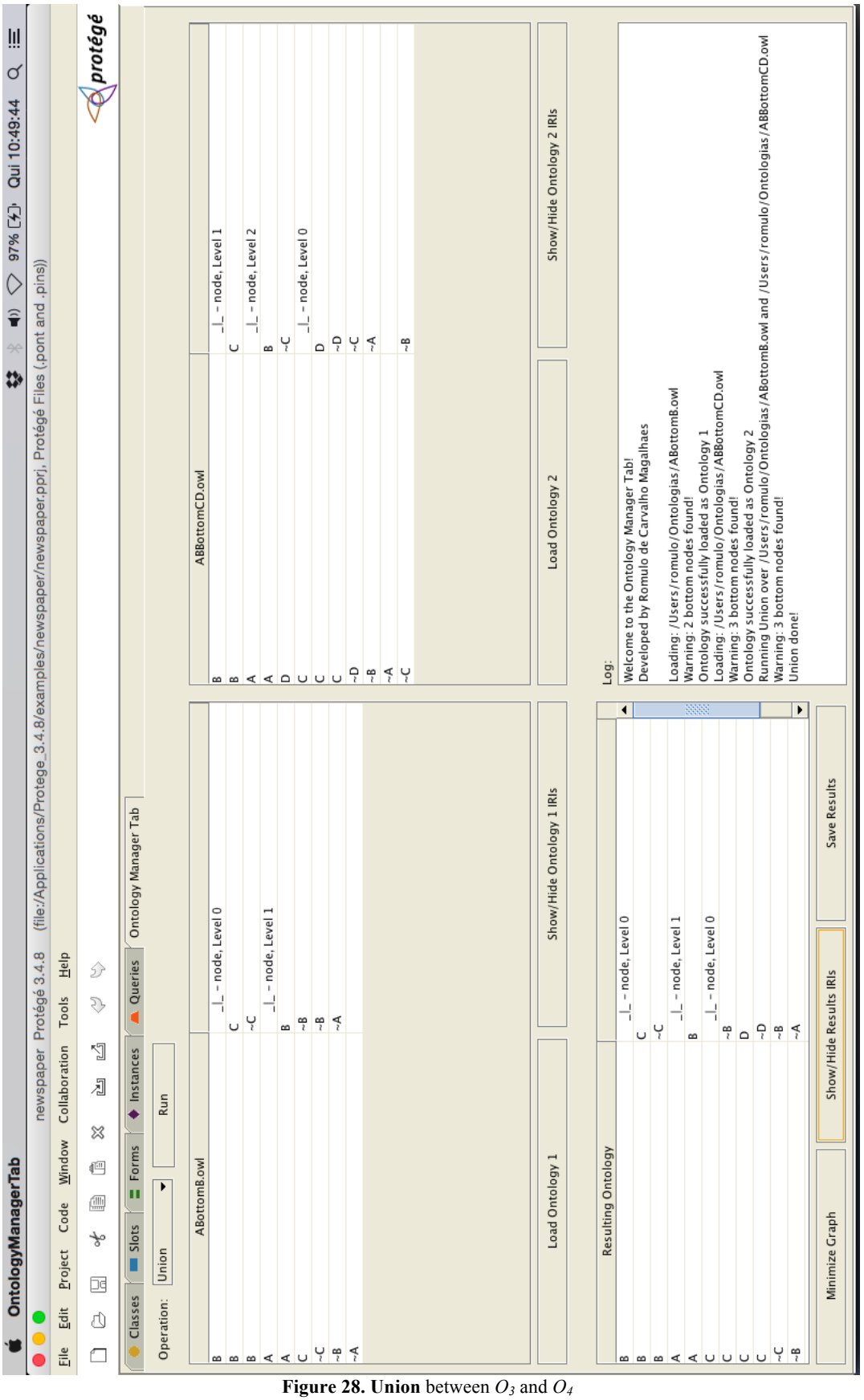

Figure 28. Union between O_3 and O_4

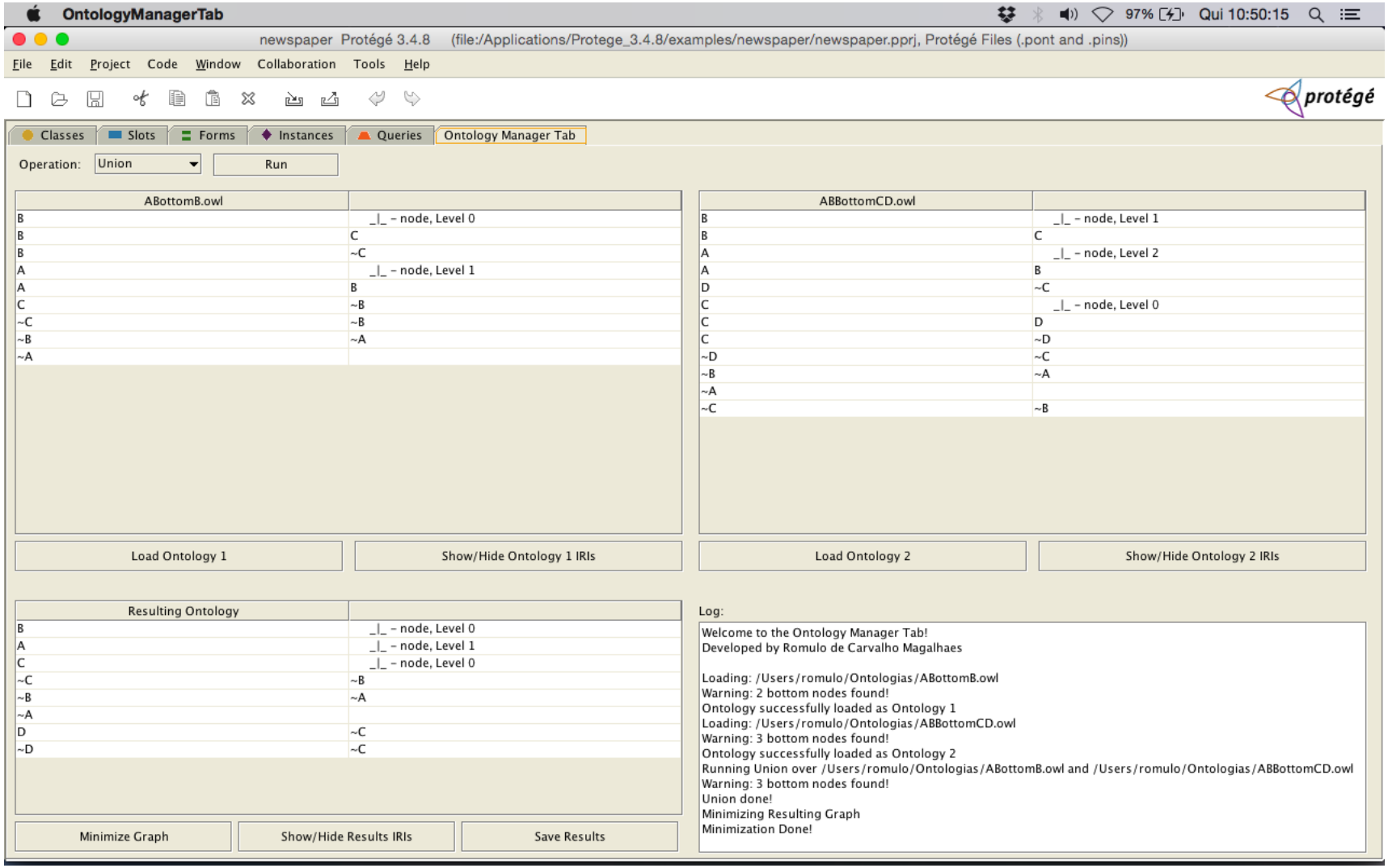

Figure 29. Graph Minimization for the **Union** between O_3 and O_4

6.4.7. Saving Resulting Ontologies

The ***OntologyManagerTab*** also provides a way for the domain specialist to save the ontologies obtained by applying the operations. The "Save Ontology" button saves the resulting ontology described in the "Resulting Ontology" table. The tool adds the string "Normalized.owl", if the name provided does not already end with it.

To illustrate this procedure, the resulting ontology for the **Projection** of FOAF ontology over V_{FF}, from **Figure 23** in Section 6.4.4, will be used. Starting from this resulting ontology, we click on the "Save Ontology" button; the file browser will be shown; and the user must choose a name for the new OWL file, as in **Figure 30,** where the name "FoafFacebookOntology" was chosen.

When choosing to save the ontology, the log will display that the resulting ontology was saved as "FoafFacebookOntologyNormalized.owl", as shown in **Figure 31.** We can now go to the folder and load the saved ontology to work on it with new functions, as shown in **Figure 32** and **Figure 33**. In **Figure 34**, we can see the recently saved "FoafFacebookOntologyNormalized.owl" file in a Text Editor application.

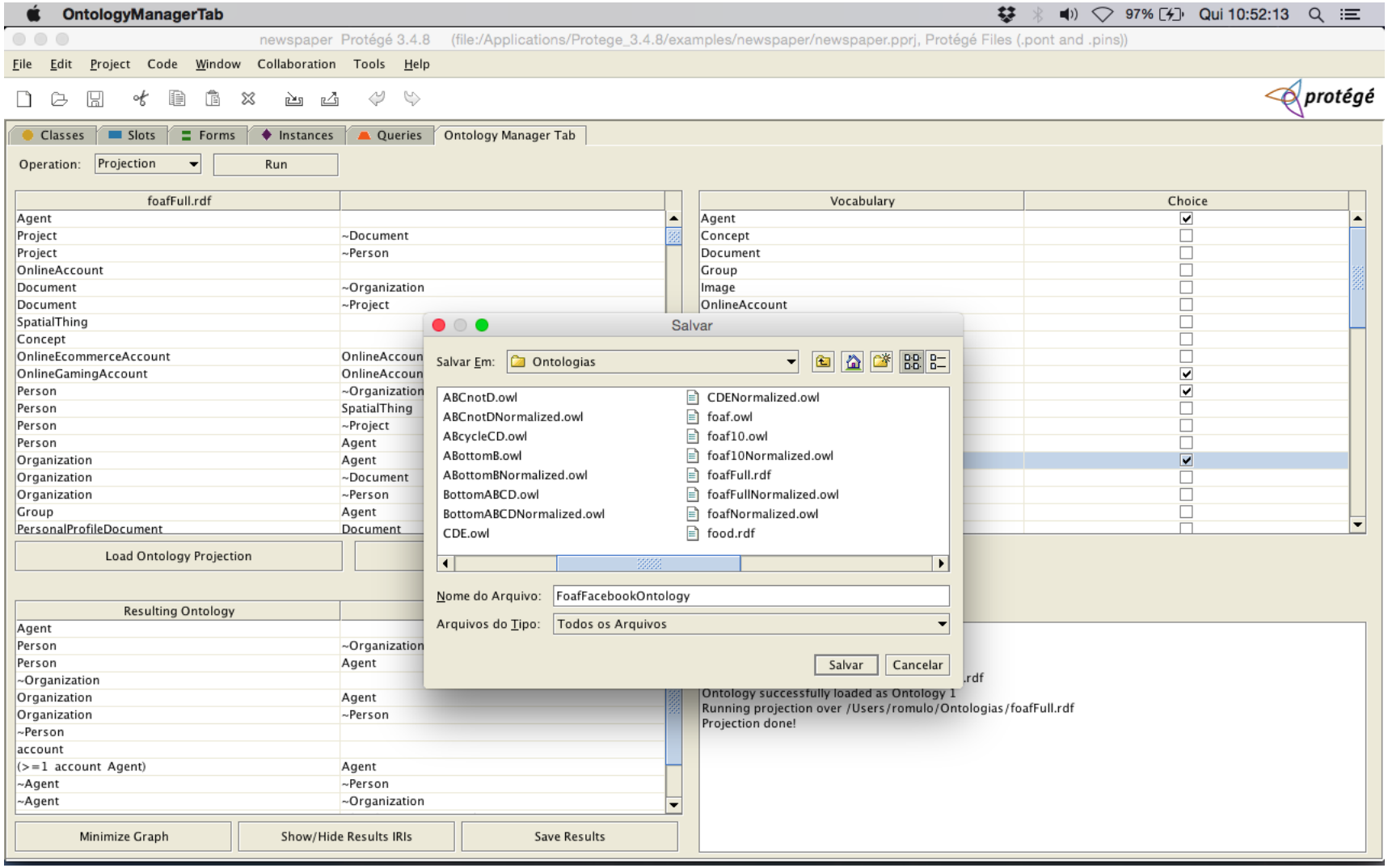

Figure 30. Saving the Resulting Ontology for the **Projection** of V_{FF} over the FOAF ontology

Figure 31. Ontology saved as "FoafFacebookOntologyNormalized.owl"

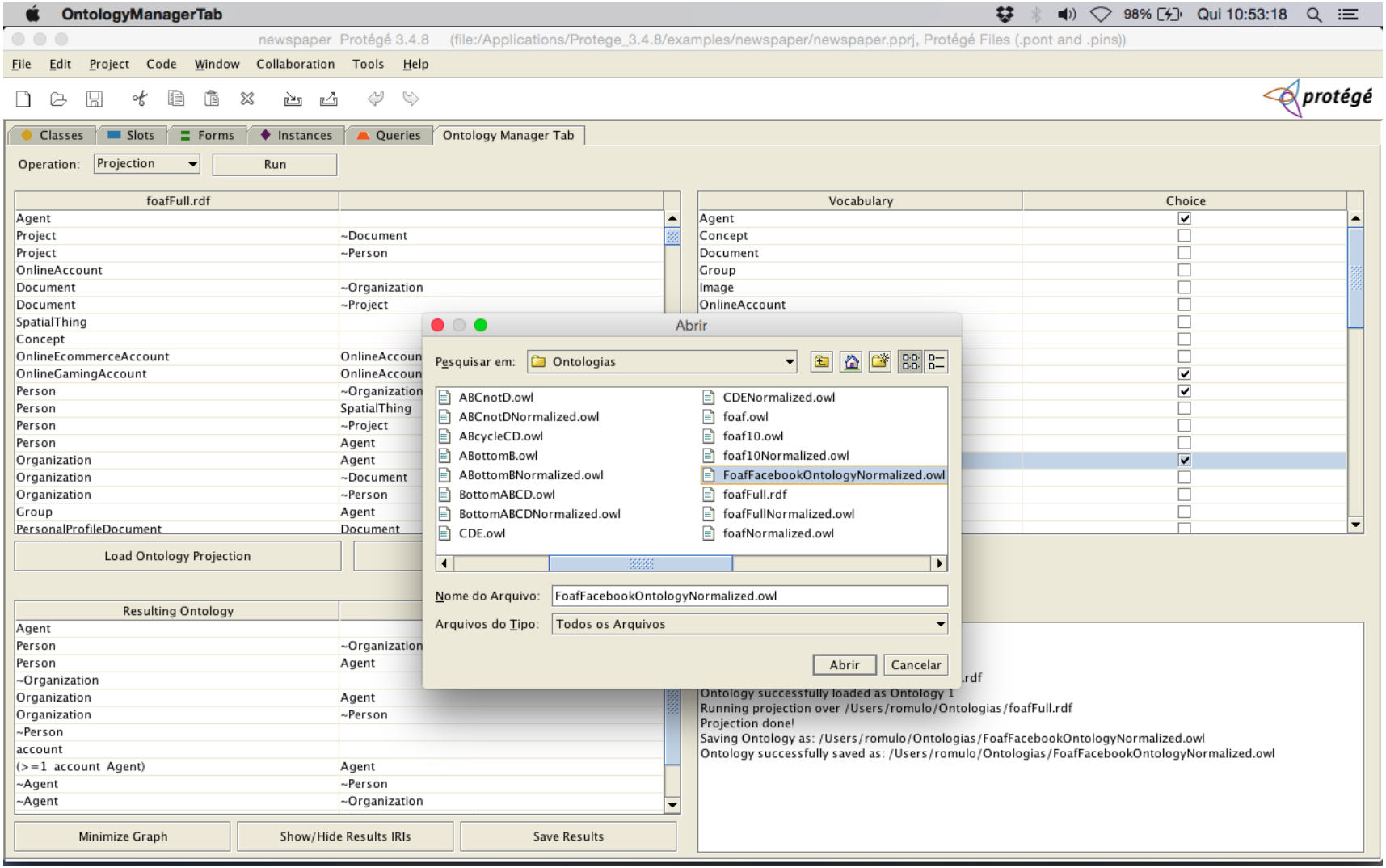

Figure 32. Loading saved ontology

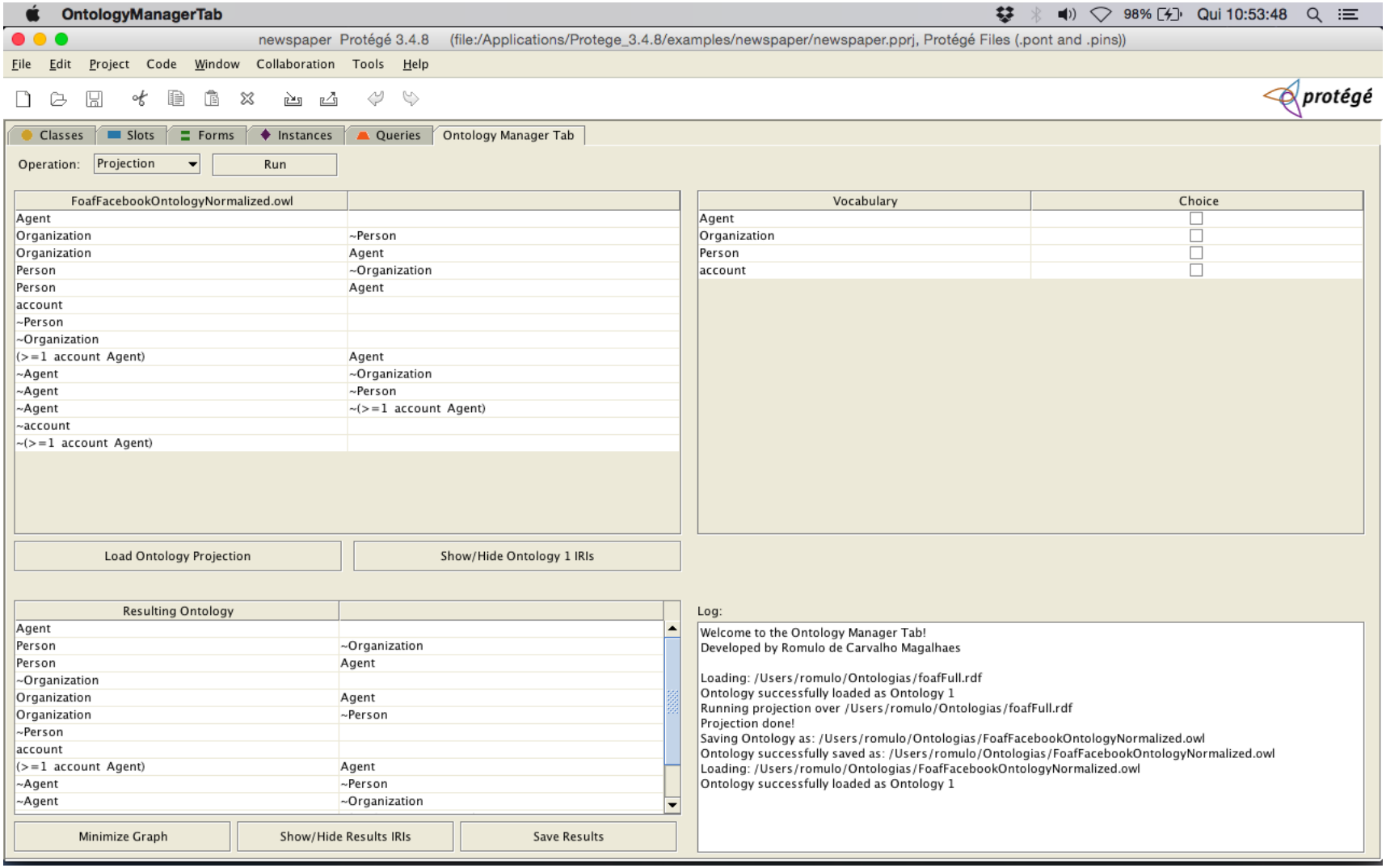

Figure 33. Recently saved ontology loaded

```
<?xml version="1.0"?>
<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2014/5/FoafFacebookOntologyNormalized#"
     xml:base="http://www.semanticweb.org/ontologies/2014/5/FoafFacebookOntologyNormalized"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology rdf:about="http://www.semanticweb.org/ontologies/2014/5/FoafFacebookOntologyNormalized"/>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Object Properties
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://xmlns.com/foaf/0.1#account -->

    <owl:ObjectProperty rdf:about="http://xmlns.com/foaf/0.1#account"/>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // Classes
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->

    <!-- http://xmlns.com/foaf/0.1#Agent -->

    <owl:Class rdf:about="http://xmlns.com/foaf/0.1#Agent"/>

    <!-- http://xmlns.com/foaf/0.1#Organization -->

    <owl:Class rdf:about="http://xmlns.com/foaf/0.1#Organization">
        <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1#Agent"/>
        <rdfs:subClassOf>
            <owl:Class>
                <owl:complementOf rdf:resource="http://xmlns.com/foaf/0.1#Person"/>
            </owl:Class>
        </rdfs:subClassOf>
    </owl:Class>

    <!-- http://xmlns.com/foaf/0.1#Person -->

    <owl:Class rdf:about="http://xmlns.com/foaf/0.1#Person">
        <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1#Agent"/>
        <rdfs:subClassOf>
            <owl:Class>
                <owl:complementOf rdf:resource="http://xmlns.com/foaf/0.1#Organization"/>
            </owl:Class>
        </rdfs:subClassOf>
    </owl:Class>

    <!--
    ///////////////////////////////////////////////////////////////////////////////////////
    //
    // General axioms
    //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->

    <owl:Restriction>
        <rdfs:subClassOf rdf:resource="http://xmlns.com/foaf/0.1#Agent"/>
        <owl:onProperty rdf:resource="http://xmlns.com/foaf/0.1#account"/>
        <owl:onClass rdf:resource="http://xmlns.com/foaf/0.1#Agent"/>
        <owl:minQualifiedCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1</owl:minQualifiedCardinality>
    </owl:Restriction>
</rdf:RDF>

<!-- Generated by the OWL API (version 3.5.0) http://owlapi.sourceforge.net -->
```

Figure 34. Generated OWL file, "FoafFacebookOntologyNormalized.owl", in a text editor

6.5 Experiments

This section describes experiments that use 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. Among the ontologies used, we can highlight: the DBpedia[5] Ontology, the FOAF[6] Ontology, fragments extracted from the MusicOntology[7], as well as the food[8] and wine[9] ontologies from the W3C OWL Guide Page[10].

[5] http://wiki.dbpedia.org/Downloads2014#dbpedia-ontology
[6] http://xmlns.com/foaf/spec/
[7] http://musicontology.com/specification/
[8] http://www.w3.org/TR/2004/REC-owl-guide-20040210/food.rdf

It is also possible to execute *chained procedures*, such as UNION(INTERSECTION(O_1,O_2), O_3), if the resulting ontology for the first operation, in this case INTERSECTION(O_1,O_2), is saved and reloaded to execute the second, UNION(O_{Int},O_3).

The experiments with all the operations reached excellent results, not only on the trivial cases but also on the experiments with the full ontologies listed before. The longest processing times were obtained with the operations using the DBpedia ontology, since its latest version generates around 4000 nodes. For this same reason, the test cases with the DBpedia ontology were also onerous to verify and validate.

All of said 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 11** shows 10 samples of processing time obtained for the operations of loading, projection over 10 random terms and union, considering the food[11] and wine[12] ontologies. All the measurements are in seconds and the last two columns show the *average* and the *standard deviation*, respectively.

Food Ontology												
Sample	**(1)**	**(2)**	**(3)**	**(4)**	**(5)**	**(6)**	**(7)**	**(8)**	**(9)**	**(10)**	**Avg**	**Dev**
Load	0,734	1,391	1,070	0,911	0,869	0,612	0,943	1,197	0,846	0,779	0,935	0,230
Projection	0,029	0,016	0,020	0,027	0,016	0,018	0,025	0,023	0,016	0,034	0,022	0,006
Wine Ontology												
Load	0,230	0,728	0,436	0,458	0,724	0,792	0,589	0,335	0,289	0,256	0,484	0,212
Projection	0,012	0,016	0,012	0,011	0,013	0,011	0,014	0,013	0,011	0,012	0,013	0,002
Union	0,409	0,439	0,348	0,480	0,367	0,403	0,438	0,416	0,380	0,436	0,412	0,039

Table 11. Experiments Processing Times in seconds

All the ontologies used in the experiments are contained in the folder "Ontologies" inside the project compressed file. If the user wants to do new experi-

[9] http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf
[10] http://www.w3.org/TR/2004/REC-owl-guide-20040210/
[11] http://www.w3.org/TR/2004/REC-owl-guide-20040210/food.rdf
[12] http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf

ments, it is recommended to use the **Projection** procedure on larger ontologies in order to extract smaller ones, that are easier to manually verify in any other tests.

6.6 Summary

In this chapter, we presented an overview of the software developed in this dissertation, called ***OntologyManagerTab*****,** describing all of its features and use examples. Section 6.2 presented the class architecture, providing a brief overview of the developed classes and how they interact. Section 6.3 explained how to compile and setup the ***OntologyManagerTab*** for use. Section 6.4 contained examples of how to use each operation in ***OntologyManagerTab***. Finally, Section 6.5 discussed the experiments executed.

7
Conclusion

7.1 Contributions

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 this end, 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 of **Projection, Union, Intersection** and **Difference** over one or two ontologies.

The ***OntologyManagerTab***, presented in this dissertation, offers these ontology operations, integrated with traditional ontology management features. The software was developed in Java as a tab plug-in over Protégé 3.4.8. However, it 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 easier to adapt as a plug-in for any other framework or as a stand-alone software.

7.2 Limitations

One of the main challenges of this work was the integration with Protégé. Despite the extensive documentation and the various guides for plug-in implementations, the information provided was very case-specific or would not work as expected. It took a combination of various guides and some experimentation to setup the environment correctly.

Another limitation of this work is related to the URI manipulation, considering the existence of two standards: *hash* URI and *slash* URI. This problem becomes more evident when an ontology uses one standard and imports terms from

another ontology that uses the other standard. Our tool must treat every OWL ontology regardless of which standard it uses, *hash* URI or *slash* URI. It became necessary to develop parsers that interpret both kinds of URI and store the terms in the constraint graph in a consistent manner. The ***OntologyManagerTab*** stores the terms without a standard, separating the term from the URI, and saves the resulting ontologies in the *hash* URI standard.

Furthermore, there was some difficulty related to the validation of the proposed operations for the more extensive ontologies, such as the DBpedia ontology, which has around 4000 terms.

7.3 Future Work

The ***OntologyManagerTab*** could include a consistency check module that locates empty classes, that is, classes which are equivalent to the bottom class. This module would guide the domain specialist in the process of removing the inconsistencies found, by showing the options of terms that could be removed, to end said inconsistency instead of just pointing them out. In the current implementation, the user is able to identify all bottom classes and find their root, but the current implementation does not guide the user in this process nor does it helps him handle them.

In addition, the interface could be more intuitive. In this dissertation, the main focus was to provide a way for the user to visualize all the information contained in each ontology, so that he could follow every term and axiom. The axiom table could be replaced by a TreeView for each ontology, with the reachable graph from each of the nodes as a tree that could be expanded in order to help the visualization. The problem with this approach might be in the heavy renderization needed for bigger and well-connected ontologies.

Furthermore, the ***OntologyManagerTab*** could be extended to execute *chained procedures* of the proposed algebraic operations in a more intuitive and direct manner, instead of requiring the user to save and load the results of the intermediary operations.

8
Bibliographical References

(Ahoy, A. V., Garey, M. R., Ullman, J. D., 1972) Ahoy, A. V., Garey, M. R., Ullman, J. D.. The Transitive Reduction of a Directed Graph. SIAM J. Comp. 1(2), 131–137 (1972)

(Antoniou, G. e van Harmelen, F., 2003). Antoniou, G. e van Harmelen, F.. Web Ontology Language: OWL. In S. Staab and R. Studer, editors, Handbook on Ontologies in Information Systems, pages 76–92.(2003)

(Artale, A. et al, 2009) Artale, A.; Calvanese, D.; Kontchakov, R.; Zakharyaschev, M. "The DL-Lite family and relations". J. of Artificial Intelligence Research 36, pp. 1–69,(2009)

(Baader, F.; Nutt, W., 2003) 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. Cambridge U. Press, Cambridge, UK. (2003)

(Berners-Lee T. *et al*., 2005) Berners-Lee, T.; Fielding, R.; Masinter, L. RFC 3986 – Uniform Resource Identifier (URI): Generic Syntax. http://tools.ietf.org/html/rfc3986 (2005)

(Berners-Lee, T., 1994) Berners-Lee, T. RFC 1630 – A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web. http://www.ietf.org/rfc/rfc1630.txt (1994)

(Berners-Lee, T., 2007) Berners-Lee, T.. Semantic Web URIs. Realizado o acesso em fevereiro de 2013, a partir de http://dig.csail.mit.edu/2007/Talks/0108-swuri-tbl/#(1) (2007)

(Berners-Lee, T., 2007) Berners-Lee, T. Linked Data: Design issues. Realizado o acesso em fevereiro de 2013, a partir de http://www.w3.org/DesignIssues/LinkedData.html (2007)

(Bizer, C. *et al.*, 2009) 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)

(Bizer, C. *et al.*, 2011) Bizer, C.; Jentzsch, A.; Cyganiak, R. State of the LOD Cloud.http://www4.wiwiss.fu-berlin.de/lodcloud/state/. (2011)

(Bleiholder and Naumann, 2006) Bleiholder, J. e Naumann, F..Conflict handling strategies in an integrated information system. In Proceedings of the IJCAI Workshop on Information on the Web (IIWeb), (2006)

(Breitman *et al*, 2006) Breitman, K., M. A. Casanova e W. Truszkowski. Semantic web: Concepts, technologies and applications (monografia da NASA em sistemas e engenharia de software). Springer-Verlag, New York, Inc., Secaucus, NJ, USA, (2006)

(Brickley,D., Miller, L., 2010) Brickley,D., Miller, L. Foaf vocabulary specification 0.98. http://xmlns.com/foaf/spec/ (2010)

(Casanova *et al.*, 2011) Casanova, M.A., Breitman, K. K., Furtado, A. L., Vidal, V. M. P., Macêdo, J.A.F.: The Role of Constraints in Linked Data. Proc. 10th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE 2011). Lecture Notes in Computer Science 7045. Springer, Heidelberg, pp. 781-799 (2011)

(Casanova *et al.*, 2012) Casanova, M.A., Macêdo, J.A.F, Sacramento, E.R., Pinheiro, A.M.A, Vidal, V.M.P., Breitman, K.K. e Furtado, A.L.. Operations over Lightweight Ontologies. OTM 2012, Part II, LNCS 7566, pp. 646–663 (2012)

(Das, S. *et al.*, 2012) Das, S., Sundara, S., Cyganiak, R. R2rml: RDB to RDF mapping language. W3C RDB2RDF working group. http://www.w3.org/TR/r2rml/ (2012)

(Gennari J. H. *et al*., 2003) Gennari, J. H., Musen, M. A., Fergerson, R., Grosso, W. E., Crubzy, M., Eriksson, H., Noy, N. F. e Tu, S.W.. The Evolution of Protege: An Environment for Knowledge- Based Systems Development. International Journal of Human-Computer Studies, 58(1):89–123, (2003)

(Goodrich, M.T. e Tamassia., R. 2001) Goodrich, M.T. e Tamassia, R.. Algorithm Design. IE-Wiley. (2001)

(Gruber, T.R. *et al*., 1993) Gruber, T.R. *et al*. A translation approach to portable ontology specifications. Knowledge acquisition, 5: 199-199 (1993)

(H. W., T. V., U. V., H. S., G. S., H. N., S. H., 2001) H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, S. Hübner.. Ontology-Based Integration of Information - A Survey of Existing Approaches. In IJCAI'01 Workshop. on Ontologies and Information Sharing (2001)

(Heath, T. e Bizer, C., 2011) Heath, T. e Bizer, C.. Linked Data: Evolving the Web into a Global Data Space. 1st. ed.[S.l.]: Morgan & Claypool, 136 p. ISBN 9781608454303. (2011)

(Hepp M. *et al*., 2008) Ontology Management: Semantic Web, Semantic Web Services, and Business Applications. Semantic Web And Beyond Computing for Human Experience. ISBN 978-0-387-69900-4. (2008)

(Hsu, H.T, 1975) An Algorithm for Finding a Minimal Equivalent Graph of a Digraph. Journal of the Association for Computing Machinery 22(1), 11–16 (1975)

(Jacobs e Walsh, 2004) Jacobs e Walsh. Architecture of the world wide web. Obtido em março de 2013. http://www.w3.org/TR/webarch/ (2004)

(Klein, M., 2002) Klein, M. et al. "Ontology versioning and change detection on the web". In Proceedings of EKAW'02, 197–212, Siguenza, Spain, (2002)

(Klyne,G. *et al.*,2010) Klyne,G.,Carroll,J.J,McBride,B. Resource Description Framework (RDF): Conceitos e sintaxe abstrata. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (2010)

(Koubková, A. e Koubek, V.,2002). Koubková, A. e Koubek, V.. Algorithms for transitive closure. Information processing letters, 81(6):289–296 (2002)

(Langegger, A.A., 2010) Langegger, A. A Flexible Architecture for Virtual Information Integration based on Semantic Web Concepts. Tese (Doutorado) — J. Kepler University Linz, (2010)

(Lenzerini, 2002) Lenzerini, M. *Data Integration: A Theoretical Perspective*. In Proceedings of ACM Symposium on Principles of Database Systems, (2002)

(M. Klein et al., 2002) M. Klein et al. "Ontology versioning and change detection on the web". In Proceedings of EKAW'02, 197–212, Siguenza, Spain (2002)

(Maedche *et al.*, 2003) Maedche et al., "Managing multiple and distributed ontologies on the Semantic WEB", VLBD Journal 12, Springer, Location, pp. 286-302 (2003)

(Manola, F. e Miller, E., 2004) Manola, F. e Miller, E.. Resource Description Framework (RDF) Model and Syntax Specification, http://www.w3.org/TR/rdf-primer/ (2004)

(McGuinness, D. L. *et al* , 2000) D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. The Chimaera Ontology Environment. In Proc. of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence, pages 1123–1124, (2000)

(McGuinness, D.L., 2002) McGuinness, D.L. .Ontologies come of age. Spinning the semantic web: bringing the World Wide Web to its full potential, page 171-192 (2002)

(McGuinness, D.L., Harmelen, F., 2010) McGuinness, D.L., Harmelen, F. OWL Web Ontology Language Overview (OWL). http://www.w3.org/TR/2004/REC-owl-features-20040210/ (2010)

(Noy, N. F. *et al*, 2004) N. F. Noy. Semantic Integration: A Survey Of Ontology-Based Approaches, ACM SIGMOD Record, vol. 33, nº4, p.65-70, (2004)

(Noy, N. F. *et al*, 2004) N. F. Noy, S. Kunnatur, M. Klein, and M. A. Musen. Tracking changes during ontology evolution. In Sheila A. Mcilraith, Dimitris Plexousakis, e Frank van Harmelen. Third International Semantic Web Conference, pages 259–273, Hiroshima, Japan (2004)

(Noy, N. F. and Musen M. A., 2000) N. F. Noy, and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In Seventeenth International Joint Conference on Artificial Intelligence AAAI/IAAI, pages 450–455 (2000)

(Noy, N. F. and Musen, M. A., 2004) N. F. Noy, and M. A. Musen. Specifying Ontology Views by Traversal. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, International Semantic Web Conference, volume 3298 of Lecture Notes in Computer Science, pages 713–725 (2004)

(Peter Haase *et al*, 2003) Peter Haase, York Sure e Denny Vrandei (Institute AIFB, University of Karlsruhe) Ontology Management and Evolution – Survey, Methods and Prototypes (2003)

(Pinheiro, A.M.A., 2013) Pinheiro, A.M.A.. OntologyManagementTool – Uma Ferramenta Para Gerenciamento De Ontologias Como Teorias Lógicas (2013)

(Rahm, E. e Bernstein,P.A. 2001) E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Mat- ching. VLDB Journal: Very Large Data Bases, 10(4):334–350 (2001)

(Ramos., J. A., 2001) Ramos, J. A..Mezcla automática de ontologías y catálogos electrónicos. Final YearProject. Facultad de Inform´atica de la Universidad Politécnica de Madrid. Spain, (2001)

(Roy,R., 1959) R. Roy. Transitivité et connexité. C.R. Acad. Sci. Paris, 249:216–218 (1959)

(Sahoo, S.S. *et al.*, 2009) 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)

(Shvaiko, P. e Euzenat., J., 2004) P. Shvaiko and J. Euzenat. A Survey of Schema-based Matching Approaches. Tech- nical Report DIT-04-087, University of Trento (2004)

(Stojanovic, L., 2004) Stojanovic, L. Methods and Tools for Ontology Evolution, Ph.D. Thesis, University of Karlsruhe, Germany (2004)

(Thompson, H. S. *et al***.,** 2004)Thompson, H. S. *et al***.** XML Schema Part 1: Structures Second Edition, W3C Recommendation. Disponível em: http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/ (2004)

(Tury, M. e Bieliková. M., 2006) Tury, M. e Bieliková, M.. An Approach to Detection Ontology Changes. In ICWE ’06: Workshop proceedings of the sixth

international conference on Web engineering, page 14, New York, NY, USA. ACM Press (2006)

(Volz R., 2003) Volz, R., Oberle, D. e Studer, R.. Implementing Views for Light-Weight Web Ontologies. In Proc. of Int. Database Engineering and Application Symposium - IDEAS, pages 160–169, Hong Kong, China. IEEE Computer Society (2003)

(Warshall, S., 1964) Warshall, S.. A theorem on boolean matrices. Journal of the ACM , 9(1):11–12 (1962)