Exploring Incremental Reasoning Approaches Based on Module Extraction

Liudmila Reyes-Alvarez[1], Danny Molina-Morales[1], Yusniel Hidalgo-Delgado[2], María del Mar Roldán-García[3], José F. Aldana-Montes[3]

[1] Centro de Informática Industrial (CEDIN), University of Informatics Sciences, Havana, Cuba
[2] Facultad 3, University of Informatics Sciences, Havana, Cuba
[3] Departamento de Lenguajes y Ciencias de la Computación, University of Malaga, Málaga, España
lreyes@uci.cu, dmmorales@xetid.cu, yhdelgado@uci.cu, {mmar, jfam}@lcc.uma.es

Abstract. This paper explores the use of module extraction for incremental reasoning of knowledge bases (KB) based on description logics (DLs). The main objective is to evaluate the different approaches that incrementally solve logical inference problems (tasks or services) based on modularization process in order to identify different strategies for implementing this process in future incremental reasoning algorithms. Three algorithms were found that use an incremental approach to solve the logical inference task of classification based on module extraction of which two are implemented and tested in this paper. The evaluation results show how the incremental reasoning based on modularization enhances the reasoning efficiency due to a modification in DLs-based KB, because this update affects only a small number of components in the KB structure.

Keywords: Module Extraction, Incremental Reasoning, Classification, Knowledge Base, Description Logic

1 Introduction and Motivation

During the evolution of the Semantic Web from 2001 until now new techniques and paradigms for knowledge representation have been proposed which contribute to integration and resource location. These resources are discovered and used not only by humans but also by computer systems. The Semantic Web facilitates the elemental organization around which a knowledge base (KB) can be created. In the literature in the area of computing, ontology terms and KB tend to be gathered together erroneously. In [1] we found a clear conceptual difference between ontology and KB as the author says:

" A shared ontology need only describe a vocabulary for talking about a domain, whereas a knowledge base may include the knowledge needed to solve a problem or answer arbitrary queries about a domain." [1]

A description logic (DL)-based KB is formed by two components described in Figure 1: 1) a vocabulary or terminology defined to represent the knowledge described in languages such as RDFS and OWL (this is known in the literature as *Tbox*) and, 2) assertions or statements about specific individuals in terms of defined vocabulary are described using RDF triples (this is known in the literature as *Abox*). Currently there are reasoners [2], [3] that solve logical inference tasks soundly and completely. The most important logical inference problem that is resolved in the KB is *Consistency Checking*, which is used to verify that the built knowledge model is consistent. Another very important inference mechanism is *Subsumption of Concepts*, which gives the user the possibility to find all sub-concepts for a given concept or to verify whether a certain concept is a sub-concept of another. This account with another inference service as *Classification*, than is to determine the appropriate location to a new concept defined in a hierarchy of concepts linked by subsumption relations. It can also be used for other inference services such as: *Logical Implication* and *Instance checking*, among others. However, the updating of a DL-based KB (i.e., addition or deletion of new knowledge) which can be very common in the *Abox* is not common in the *Tbox*, because the terminology defined does not change frequently over time, unless a terminological evolution in the represented domain occurs. In the *Abox*, the represented knowledge is more transitory because it is composed by clear concepts that make up the *Tbox*, therefore it is modified more frequently over time.

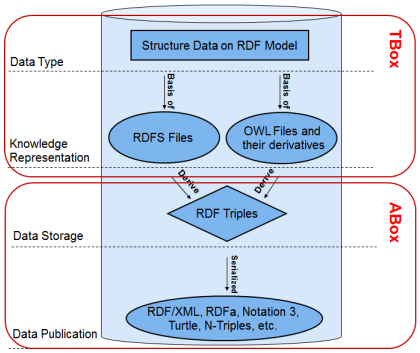

Fig. 1. Components of a DL-based KB on the Semantic Web.

Several researchers have studied, analyzed and compared the multiple semantic reasoners that exist today (e.g., [2]and [3]) achieving results which point to the need to define reasoning algorithms to infer knowledge on DLs-based KBs and, the absence of reasoners for efficiently solving the logical inference problems given a small modification in the DLs-based KBs. This is because the reasoners repeat all the steps of the reasoning process from scratch each time a change/update is made in the represented knowledge, that makes up the DL-

based KB, inefficient aspect, taking into account that the change brings with it two main conditions:

Condition 1. Always before the update there needs to be a reasoned version of the KB.

Condition 2. Normally, a small updating of the KB does not affect all the elements of its structure.

The solution to this problem is known in the literature as *incremental reasoning*. Researchers have attempted to provide a solution for the aforementioned problem but only using one of the two conditions. It restricts the solution to only a small part of the problem. A first approach is to adapt the algorithms defined for reasoning in the Semantic Web and treat them as static DL-based KB [4]. Another approach establishes improved stream reasoning techniques [5], [6], [7], [8]. Finally, the volume of data that is expected to be stored in our KB is very large. Therefore, neither of these techniques by themselves help us to completely solve the problem to ensure scalability of reasoning. Much research is needed in this area, although the results obtained in some cases based on materialization or modularization methods are encouraging and provide a basis for future work.

From among the methods that have been used to solve a part of the aforementioned problem based on the second condition, we find the modularization of KB. This is responsible for separating the KB into small portions. This is obviously because it is easier to apply reasoning tasks on small portions of the KB than its whole. Therefore, incremental reasoning of KBs is particulary relevant when exploring investigations based on modularization. Taking into account that each correction made in the KB, affects only a part of its structure, where *module extraction* plays a relevant role.

In this paper we analyze the semantics of DL reasoning mechanisms based on the modularization process with the goal of making it easier to identify the points at which researchers can contribute to the practical development of the Semantic Web. Therefore, this paper describes an evaluation of proposed algorithms to solve the *Classification* logical inference problem based on module extraction during the reasoning process following an incremental approach. The document is organized as follows: Section 2 introduces incremental reasoning algorithms. Section 3 explains the workflow followed to test the algorithms. Section 4 displays the evaluation results. Finally, Section 5 presents the conclusions and direction of future research.

2 Incremental Reasoning Algorithms

After completing a review of the state of the art in KB reasoning procedures that support incremental reasoning based on module extraction (or modularization), were studied and identified. We found only three reasoning processes based on modularization [4],[9], [10],[11]. These procedures solve the inference problem of *Classification* by following an incremental approach based on module extrac-

tion. These algorithms follow different extraction philosophies but all resolve the *Incremental Classification* problem.

The three algorithms that have been studied are: 1)incremental classification for ontologies in the SROIQ language [9], [10], 2) incremental classification for ontologies in the EL+ language [11], [10] and, 3) incremental classification by the Pellet reasoner[4]. When we looked at the implementation conditions of the incremental classification algorithm for ontologies in the *EL+* language [11], [10], we found that in order for it to be implemented it is necessary to modify the internal structure at reasoner, which limits its use by user. We have therefore decided not to implement this algorithm for its evaluation in the work presented here.

For a clearer understanding of the algorithms tested the reader needs to be aware of the following definitions: **semantic location** and **signature**.

*A **signature** is any subset **S** of **R**⊎**C**⊎**I**. The signing of an axiom α is the set **Sig**(α) of atomic roles **R**, atomic concepts **C**, and individuals **I** that occur in α. Ontology signature **Ont** is set **Sig(Ont)** for symbols that occur in **Ont**.*[9],[10]

***Semantic Location**: Let **S** be a signature. It is said that an interpretation **I** is local for **S** if for each atomic concept **A**/∈**S** and each atomic role **R** that does not belong to **S**, it has **AI**=**RI**=ø. A SROIQ axiom α is semantically local for a signature **S** if **I**|=α for all **I** that is local for **S**. A SROIQ ontology **Ont** are local for **S** if each axiom in **Ont** is local to **S**.*[9],[10]

2.1 Module Extraction and Incremental Classification for SROIQ

The modularization algorithm proposed by Bernardo Cuenca and his colleagues [9], [10] is based on the definitions specified in the fourth section of his paper [10]. It was applied to ontologies described through the *SROIQ* language for the knowledge representation. The algorithm is implemented given an ontology in the *SROIQ* language and a signature. Internally this algorithm uses a local sub-routine (α, **S**) to check whether an axiom α is semantically local in **S**. This obtains as output, a module that belongs to the ontology for the given signature. These modules contain the axioms that are not semantically local for the signature.

The aforementioned algorithm is used for solving the logical inference problem of *classification* following an incremental approach. Given an ontology **Ont1** and a change $\triangle$**Ont1** = ($\triangle$-**Ont1**, $\triangle$+**Ont1**), which consists of the axioms set that is removed and added to **Ont1**. The algorithm computes partial subsumptions $\sqsubseteq$2 for the ontology resulting **Ont2**= (**Ont1**\$\triangle$-**Ont**)$\bigcup\triangle$+**Ont1** by reusing the subsumption relation $\sqsubseteq$1 as calculated in **Ont1**. In order to perform this operation, the algorithm internally keeps modules **Ont1A** and **Ont2A** for each atomic concept **A** and modules **Ont1**$_\top$ and **Ont2**$_\top$ for the empty signature. The algorithm consists of the following phases:

Phase 1: Processing of new symbols.

Ont1A modules and partial order subsumption $\sqsubseteq$1 for **Ont1A** is extended for all atomic concepts **D** recently introduced. **D** module (about which no information has been given yet) is equivalent to the empty signature module **Ont1**$_\top$.

Thus, there are: **Ont1D**=**Ont1**$_\top$, **Ont1**|=**D**$\sqsubseteq$**B** if and only if **Ont1**|=$_\top\sqsubseteq$**B**, **Ont1**|=**A**$\sqsubseteq$**D** if and only if **Ont1**|=**A**$\sqsubseteq_\bot$.

Phase 2: Identification of affected modules.

The sets **M-** and **M+** containing the **A**$\in$**CN**$_\top$(**Ont1**) for the corresponding modules must be modified by removing and/or adding axioms. If α is removed from **Ont1** and not local (**Sig(Ont1A)**) then α must be removed from **Ont1A**. If α is added to **Ont1** and is not local (**Sig(Ont1A)**). Then **Ont1A** needs to extend at least α.

Phase 3: Calculation of new modules and subsumptions.

The affected modules that have been found in the previous phase are re-extracted and the others are copied. Then each subsumption A$\sqsubseteq$B, using *Proposition 1* in Section 4.1 of the paper presented in [10], or recalculates the module against **Ont2A** or a reused one from **Ont1**.

2.2 Module Extraction and Incremental Classification by the Pellet Reasoner

The algorithm for module extraction in the Pellet semantic reasoner is implemented virtually three times.

The first implementation extracts the imports closure module of an ontology **Ont** given by the determined signature **S**. The modules contain axioms ed with signature elements that describe how they relate to each other. The reasoner has a method for closing the imports and returns the ontologies set $\triangle$**Ont** related with the current ontology that contains axioms relevant to the given signature.

The second implementation for the module extraction implemented virtually, receives as input parameters, an ontologies set $\triangle$**Ont**, which contains the original ontology imports closure, a signature **S** and a variable that defines the module type ***ModuleType***. It has already implemented subroutines ***Extract_ Top_ Module*** (axioms, **S**) and ***Remove_ Bot_ Module*** (axioms, **S**), which extract the axioms set that is relevant to the given signature calling the third algorithm implemented virtually: ***Remove_ Module*** (axioms, **S**, ***LocationType***) according to the module type the variable ***ClassLocation*** is created. The algorithm returns the closing imports module, which is closely related to the given **S** signature.

There are four module types supported: 1) lower module ***TOP*** has subclasses of signature elements, 2) upper module ***BOTTOM*** contains super-classes of signature elements, 3) top of the lower module ***BOTTOM_ OF_ TOP*** to extract the top module of lower module and, 4) lower half of the upper module ***TOP_ OF_ BOTTOM*** to extract the lower module of the upper module.

The third virtual implementation for the module extraction receives as parameters: an axioms set, a signature **S**, and a ***ClassLocation*** variable, to create a ***ModuleExtractor*** class instance which is defined in the reasoners′ sources. Once all the axioms have been added to the ***Extractor_ Module*** instance, it invokes another method of ***Extractor_ Module***(**S**), which receives only a signature **S**. This method ***Extractor_ Module***(**S**) belongs the ***ModuleExtractor*** class. The method returns an axioms set that is related to the given signature.

The incremental classification strategy proposed by Pellet does the following: for all modules that are affected, collects all the axioms and classifies all at once. This allows the exploitation of current classification optimizations. In addition it leverages the classification carried out earlier which is in the cache of the reasoner. The following describes in detail each step of the algorithm.
Phase 1: Collects the entities whose modules are affected. Collects entities affected by the addition of axioms and, collects entities affected by deletion of axioms.
Phase 2: Creates an ontology for all axioms of all affected modules.
Phase 3: Loads the removed module in a new instance of the reasoner.
Phase 4: Classifies the module. It builds a class hierarchy for the new instance of the reasoner created in phase three which loads the module that was removed. It creates a set of empty classes. For each entity in the entity set, which contains the affected modules, if the entity is an instance of a class then it adds the entity as a class, the whole class created in the previous line. Finally, the class hierarchy of the cache of the reasoner is updated with the new changes.

The tricky part of this algorithm is the time it takes to create a new instance of the reasoner, once it has created this instance, from scratch it then starts the classification task for the knowledge base of that instance, also from scratch, which in this case would be all affected modules identified in the previous lines. This may considerably delay the reasoner′s response time.

3 Workflow to Run Evaluations

We have implemented a workflow to develop and test the algorithms. The workflow consists of six phases.
Phase 1. Creating the infrastructure to enable the assessment of reasoning algorithms based on modularization. The infrastructure created consists of three parts: 1) selecting the Netbeans framework to implement and test the algorithms, 2) importing the project created from the OWL API libraries dedicated to managing ontologies in the Pellet semantic reasoner for successful implementation and, 3) identifying the non-functional requirements of the machine where the algorithms′ assessments will run.
Phase 2. Through out Phase 1 the biomedical ontologies are loaded into the NetBeans from a local web server. Element allows us to work with the ontologies to modify and access these through the source code in the framework.
Phase 3. After completion of Phase 2 the implementation of the module extraction algorithms for the incremental Classification an ontology is done. These are the module extraction algorithm and the incremental classification for SROIQ ontologies [10] [9] and, the module extraction algorithm and incremental classification utilized by the Pellet reasoner in version 2.3.0 [4].
Phase 4. The tests are executed upon the completion of Phase 3 on biomedical ontologies such as Gene Ontology, NCI-Thesaurus and Galen. First, the correct implementation of the algorithms is checked. Second, the response times between

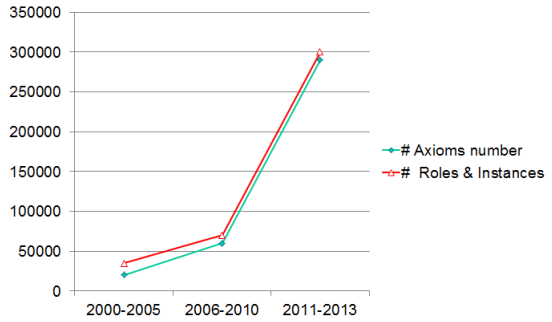

Fig. 2. Ontologies have been increasing in size and complexity over the years.

the algorithms and the necessary computational memory cost are compared, which are the parameters measured in this study.

Phase 5. Then, the algorithms implementation of incremental classification [9] [10] [4](which are described in Section 2) based on any of the modularization processes implemented in Phase 3, is performed after the successful completion of Phase 4.

Phase 6. After Phase 5, the incremental classification algorithm based on the modularization processes described in Section 2 are tested. We check the response times and the cost of the computer memory required, which are the variables of interest for this study.

The biomedical ontologies (e.g., Gene Ontology, NCI Thesaurus) that have been used to run the evaluation have increased in size and complexity over the years (see Figure 2). Hence, the axioms number, roles and individuals that the algorithms had were smaller than when they were first implemented and tested by the original authors.

4 Evaluation Results

The following aspects should be taken into account when comparing the results obtained by the original authors with those obtained with the implementation carried out in this paper: 1) biomedical ontologies were 40% smaller than their current size which has undergone a considerable increase and, 2) the non-functional requirements where the original algorithms were implemented perform better now. Although in both cases the results are satisfactory, our implementation execution time is slightly higher.

The evaluations are performed through the Pellet semantic reasoner (version 2.3.0) based on OWL DL, which implements a procedure based on Tableaux. The evaluation of the procedures described in the second section did not change the reasoner internally. Moroever we used the OWL API of the Pellet reasoner via the NetBeans framework that implements the algorithms in the Java language. The evaluations were done on a computer with the following features: operative

system is Windows 7 32-bit, RAM memory is 1.0 GB, the processor is Intel Dual Core 3.00 GHZ and hard disk capacity is 80 GB.

For the evaluations a well-know ontologies set is selected as we show in Table 1. In this table we provide basic information about the ontologies to be evaluated, that include the language in which they are expressed, the atomic concepts number and axioms that contained. First we show the evaluation results of the

Table 1. Ontologies information to assess.

Ontology	**DL Language**	**Atomic Concept**	**Axioms**
Galen	*ALEH+*	23141	62179
Gene Ontology	*AL+*	20465	89370
NCI Thesaurus	*ALE*	27652	422953

module extraction procedures and then the results of the incremental reasoning algorithms, both described in Section 2. Tables 2 y 3 provide information on the results of the evaluation such as: ontology that we evaluated, signature for the module extraction, number of axioms in the module, module extraction time and finally, in the last column of the table we show that the number of modules extracted in this case is one, because it was done manually.

Table 2. Evaluation of the Module Extraction for the SROIQ language.

Ontology	**Signature**	**# Axioms**	**Extraction time (seconds)**	**Modules**
Galen	*Heart Liver BloodPressure*	40	174	1
Gene Ontology	*GO_0043234 GO_0042946 part_of*	3	360	1
NCI Thesaurus	*Findings_and_Disorders_Kind Inflammation Epilepsy*	19	270	1

Table 3. Evaluation of the Module Extraction by the Pellet Reasoner.

Ontology	**Signature**	**# Axioms**	**Extraction time (seconds)**	**Modules**
Galen	*Heart Liver BloodPressure*	65	199	1
Gene Ontology	*GO_0043234 GO_0042946 part_of*	154	450	1
NCI Thesaurus	*Findings_and_Disorders_Kind Inflammation Epilepsy*	38	280	1

In Tables 2 and 3 we show that there is a remarkable difference in the axioms affected given the same signature and the same ontology as inputs. This is given for the difference in the performance of both algorithms, that has nothing to do with each other, are totally different. The modularization procedures proposed by Bernardo and his collaborators [10] works on the axioms of the ontology directly, while the proposed modularization by Pellet works with ontology imports (the ontology is divided into smaller ontologies and then analyzes each ontology individually) to build the module.

Then, incremental classification procedures are evaluated. Tables 4 and 5 provide basic information about the ontology to which each procedure is applied. Column 1 indicates the ontology evaluated. Column 2 indicates the number of axioms that are removed and added to the ontology for the evaluation **n=1,2,4**. Column 3 shows number of the affected modules. We can see that for large ontologies only a very small number of modules are affected by the given update. The values obtained are correlated with the percentage of positive subsumption relations in the ontology. Column 4 shows the time (in seconds) taken to locate, remove and re-classify the affected modules. Column 5 shows the total time for the union of the affected modules. The aim of the assessments is to simulate the

Table 4. Incremental Classification for the SROIQ Language. Time in seconds.

Col. 1	**Col. 2**	**Col. 3**	**Col. 4**	**Col. 5**
Ontology	**n**	**# Modules affected**	**Modules Re-classification**	**Time**
Galen	1	1114	50	260
Galen	2	988	63	170
Galen	4	662	57	197
Gene Ontology	1	425	40	390
Gene Ontology	2	358	30	400
Gene Ontology	4	793	40	320
NCI Thesaurus	1	125	93	480
NCI Thesaurus	2	94	85	392
NCI Thesaurus	4	287	73	510

ontologies′ evolution process when **n** axioms are modified (each of these changes can be seen as a simultaneous removal and/or addition of an axiom).

5 Conclusions and Direction of Future Research

This approach succeeds in creating an infrastructure to evaluate incremental reasoning algorithms that do not modify the internal structure of a semantic reasoner. When we compare the results achieved by the original authors and our implementation, we conclude that the implementation carried out in this paper is correct. Given the size parameters of the ontologies and non-functional requirements which were assessed, we can easily see that the original authors

Table 5. Incremental Classification by the Pellet Reasoner. Time in seconds.

Col. 1	Col. 2	Col. 3	Col. 4	Col. 5
Ontology	**n**	**# Modules affected**	**Modules Re-classification**	**Time**
Galen	1	253	67	294
Galen	2	192	71	197
Galen	4	83	72	240
Gene Ontology	1	236	68	392
Gene Ontology	2	479	46	420
Gene Ontology	4	247	90	370
NCI Thesaurus	1	354	110	490
NCI Thesaurus	2	215	99	397
NCI Thesaurus	4	193	86	560

had higher performing computers than ourselves and the ontologies at that time were smaller (see Figure 2).

We therefore reach the following conclusions:

First, in the incremental classification algorithm for SROIQ ontologies [10], the reasoner is used only as a " black box" to answer queries of subsumption. This prevents any modification to the reasoners internal structure which in turn means that any reasoner based on OWL (and its derivatives) can be used.

Second, the virtual algorithms of modules extraction used by the Pellet semantic reasoner [4] can be very complex when reasoning tasks are applied to large ontologies. As a consequence of this hundreds of imports closure can be generated. After obtaining these imports (which are generated only in the original ontology), the algorithms analyze all the axioms contained within them.

Third, the evaluation results suggest that the incremental reasoning approach based on modules extraction is especially useful in large ontologies with DL basic languages, such as $EL+$ and AL.

Summarizing, both procedures are accurate in the extraction of modules and do not produce errors, such as: removing an empty module, the paralyzation of its subroutines, and the data returned at an inaccurate time. However, the superiority of the incremental classification procedure for ontologies in the SROIQ language is clear in terms of response times and the main memory usage we see in the evaluations.

Future research will consist of developing a model of incremental reasoning on instance assertions based on module extraction and materialization using non SQL databases. The algorithms of the model will be tested using the infrastructure that has been created. This model will be implemented and tested in the DBOWL reasoner [12], developed by the KHAOS group at the University of Malaga. It is a true complement to current OWL reasoners and it is very important because it is able to support ontologies with much bigger *Aboxes* than traditional systems based on description logic and satisfiability. This is especially significant for some applications, such as life sciences, where particularly large ontologies are used. The DBOWL evaluation results described in [13] will

enable the prediction of the behavior and trend of logical inference services that are solved by *Abox* reasoning in a DL-based KB. However, although DBOWL is a non-materialized approximation because the reasoning is implemented using SQL views, it is able to deal with all types of namespaces, to know when a class is non-satisfiable and to check the ontology consistency in this case.

Acknowledgements

This work is partially supported by the Spanish MEC Grant (TIN2008-04844, TIN2011-25840) and by the Andalusian Regional Government Grant (P07-TIC-02978, P11-TIC-7529).

References

1. Gruber, T.R., et al.: A translation approach to portable ontology specifications. Knowledge acquisition **5**(2) (1993) 199–220
2. Mishra, R.B., Kumar, S.: Semantic web reasoners and languages. Artificial Intelligence Review **35**(4) (2011) 339–368
3. Dentler, K., Cornet, R., ten Teije, A., de Keizer, N.: Comparison of reasoners for large ontologies in the owl 2 el profile. Semantic Web **2**(2) (2011) 71–87
4. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics: science, services and agents on the World Wide Web **5**(2) (2007) 51–53
5. Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Incremental reasoning on streams and rich background knowledge. In: The Semantic Web: Research and Applications. Springer (2010) 1–15
6. Barbieri, D., Braga, D., Ceri, S., Valle, E.D., Huang, Y., Tresp, V., Rettinger, A., Wermser, H.: Deductive and inductive stream reasoning for semantic social media analytics. Intelligent Systems, IEEE **25**(6) (2010) 32–41
7. Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: Ep-sparql: a unified language for event processing and stream reasoning. In: Proceedings of the 20th international conference on World wide web, ACM (2011) 635–644
8. Barbieri, D.F., Braga, D., Ceri, S., VALLE, E.D., Grossniklaus, M.: C-sparql: a continuous query language for rdf data streams. International Journal of Semantic Computing **4**(01) (2010) 3–25
9. Grau, B.C., Halaschek-Wiener, C., Kazakov, Y.: History matters: Incremental ontology reasoning using modules. In: The Semantic Web. Springer (2007) 183–196
10. Grau, B.C., Halaschek-Wiener, C., Kazakov, Y., Suntisrivaraporn, B.: Incremental classification of description logics ontologies. Journal of Automated Reasoning **44**(4) (2010) 337–369
11. Suntisrivaraporn, B.: Module extraction and incremental classification: A pragmatic approach for el + ontologies. In: The Semantic Web: Research and Applications. Springer (2008) 230–244
12. Roldan-Garcia, M.M., Aldana-Montes, J.F.: Dbowl: Towards a scalable and persistent owl reasoner. In: Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on, IEEE (2008) 174–179

13. Roldan-Garcia, M.M., Aldana-Montes, J.F.: Evaluating dbowl: A non-materializing owl reasoner based on relational database technology. In: OWL Reasoner Evaluation Workshop (ORE 2012). (2012)