HyLAR+: Improving Hybrid Location-Agnostic Reasoning with Incremental Rule-based Update

Mehdi Terdjimi
Université de Lyon
Université Lyon 1 - CNRS
LIRIS UMR5205
F-69622, Lyon, France

Lionel Médini
Université de Lyon
Université Lyon 1 - CNRS
LIRIS UMR5205
F-69622, Lyon, France

Michael Mrissa
Université de Lyon
Université Lyon 1 - CNRS
LIRIS UMR5205
F-69622, Lyon, France

firstname.lastname@liris.cnrs.fr

ABSTRACT

Web applications that rely on datasets of limited sizes to handle small but frequent updates and numerous queries have no simple way to define where data should be stored and processed. We propose a reasoning framework that can be integrated in Web applications and is able to perform the same reasoning tasks on both client or server sides. This framework embeds a rule-based reasoning engine that uses an algorithm relying on both incremental reasoning and named graphs. We evaluate the performance of our approach and compare the effects of incremental reasoning and named graphs in different experimental conditions. Results show that our reasoner can significantly reduce response times to INSERT and DELETE queries. During the demo we will exhibit how it can be used to perform reasoning tasks based on client-generated information and improve Web applications with location-agnostic reasoning.

General Terms

Design, Performance, Experimentation

Keywords

semantic web, reasoning, client-side reasoning, rule-based reasoning

1. INTRODUCTION

More and more Web applications generate and use datasets of limited sizes. This is the case in domains such as personal small data [2] or Web of Things [3], and more generally for applications that keep track of individual user's activities to adapt their behavior. Such datasets are subject to small but frequent updates and numerous queries. However, depending on the data volumes, process complexity and client capabilities, there is no simple way to define where data should be stored and processed and make appropriate choices at design time. Especially when semantic reasoning comes into play, the diverse reasoning tasks involved in bootstrapping a reasoner (classification), updating data (insert, delete and re-insert) and querying (select) must be taken into account wrt. the client processing capabilities. To partly cope with this issue, we designed and proposed the Hybrid Location-Agnostic Reasoner (HyLAR) architecture [8]. HyLAR is a JavaScript-based reasoning framework that can deploy different parts of a reasoning process either on the server or on the client.

In [8], we found out that in some cases, ontology classification could be performed on the server and the classified ontology could be shipped to the client so that querying could be performed locally. However, the limited OWL2-EL reasoner embedded in our HyLAR framework was only designed to perform these two tasks and data update was not taken into account. In this paper we propose an improvement of the HyLAR framework. It relies on an OWL2-RL reasoner that is able to perform (re)-insert and delete operations, thus providing a wider range of use to this framework. Moreover, this reasoner includes optimizations such as incremental reasoning [4] and named graphs support.

The paper is organized as follows. In Section 2, we review related work on reasoning optimization techniques and adaptive reasoning. We present our approach together with its implementation in Section 3 and our evaluation in 4. We summarize and discuss our results to give guidelines for future work in Section 6.

2. RELATED WORK

2.1 Reasoning in OWL profiles

OWL 2 profiles[1] allow adjusting the trade-off between expressive power and efficiency. Each profile favorizes specific reasoning tasks in terms of complexity [7]. EL is suitable for very large TBoxes, such as in biomedical ontologies and performs well with ontology consistency, class expression subsumption and instance checking. QL is appropriate for applications that manipulate high volumes of instances and targets query answering. RL relies on rule languages to offer a reasonable trade-off between expressive power and scalable reasoning for ontology consistency, class expression satisfiability, class expression subsumption, instance checking and conjunctive query answering [7]. As instance retrieval is the

[1] http://www.w3.org/TR/owl2-profiles/

WWW'16 Companion, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4144-8/16/04.
http://dx.doi.org/10.1145/2872518.2890542 .

most important inference task in RL [6], most RL reasoners pre-compute and explicitly store inferences, so that queries can be answered by querying the knowledge base (KB) [5]. One optimization of OWL2-RL reasoners is the Incremental Reasoning (IR) approach presented in the DRed algorithm from [4, Section 7]. It allows avoiding recalculating all consequences when partly updating the KB. To do so, facts in the KB are tagged as explicit (when directly inserted in the KB) or implicit (when derived from rules). At update time, the algorithm first performs an over-deletion step, to remove all explicit facts that are stated to be removed in the query, along with their inferred consequences. Then a rederivation step deduces all consequences of newly added explicit facts and inserts both new explicit and implicit facts in the KB. In order to reduce computation times, the algorithm performs these two steps over a limited selection of rules in the causes of which the explicit facts to be deleted/inserted appear.

2.2 Reasoning distribution

Several studies have been conducted to optimize reasoning tasks, from classical model in which all the semantic data are gathered, cleaned, lifted and stored on the server side to full client-side reasoning. On the server side, SPARQL federated queries provide a solution to handle the query answering step. The ANAPSID [1] query engine relies on several endpoints in order to split the data, decompose queries and select appropriate sources. The Triple Pattern Fragments (TPF) [9] approach is based on a principle close to federated querying but involves the client in the answering process. It allows "intelligent clients" to query TPF servers for triples that match a particular triple pattern and therefore reduce endpoints workload.

While several embedded reasoners exist to target mobile applications, most of them have been developed and optimized to fit certain languages and domain needs. To the best of our knowledge very few client-side reasoners are available for Web applications. EYE[2] is a NodeJS[3]-compatible reasoner capable of inferring on FOL rules. EYE performs server-side reasoning and has not been ported onto the client side. Based on the JSW framework, OWLReasoner[4] allows client-side processing of SPARQL queries on OWL2-EL ontologies. It can perform ontology classification and converts the Tbox and Abox into an internal relational database. SPARQL queries sent to the reasoner are rewritten into SQL queries and processed on the database. Its SPARQL engine is limited to basic rule assertions. HyLAR [8] is a framework that allows hybrid OWL2-EL reasoning. That means that both server side (Node.js) and client side (AngularJS[5]) can execute reasoning tasks using the same parts of code. Reasoning in HyLAR relies on the OWLReasoner engine.

3. HYLAR+

We propose HyLAR+[6], a new version of the HyLAR framework. We refactored its architecture and made it embed a JavaScript OWL2-RL reasoner that enables rule-based transformation, incremental reasoning and named graphs support. The HyLAR+ architecture is depicted in Figure 1 and comprises two main components: a parser and a reasoner. Both of them have been reimplemented to be faster and provide more reasoning capabilities (*i.e.* more constructs) than those of the JSW framework. The parser prepares the statements for the reasoner. It can now take two kinds of inputs: RDF/XML ontology files and SPARQL queries (INSERT DATA, DELETE DATA or SELECT). This allows bootstrapping the reasoner with an initial ontology, updating its contents when data sources change and querying it. For the needs of our evaluation, the reasoner embeds two different algorithms: a regular (a.k.a. "greedy") algorithm that processes all available rules on the whole KB at each request and the IR algorithm described in Section 2.1. Each reasoning task can be performed with any of these two algorithms. The rule base is divided into OWL2-RL (see below) and domain-specific rules. The KB stores facts (*i.e.* triples) in a local relational database and converts SPARQL request into SQL using Trim Path[7].

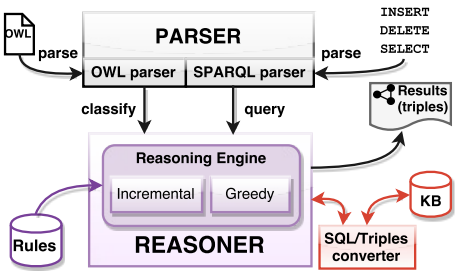

Figure 1: General architecture of HyLAR+

3.1 RL partial support and constructs

Our reasoner currently supports partial OWL2-RL semantics. The currently implemented rules are shown in Figure 2. In addition to the previously supported class subsumption and Object properties, HyLAR+ Parser and Reasoner modules now support Datatypes and Datatype properties, as well as additional OWL constructs: ObjectSomeValuesFrom, DataSomeValuesFrom, InverseObjectProperties, ObjectPropertyRange, ObjectPropertyDomain and DataPropertyRange. More rules can be added depending on the expressivity level needed by the application.

3.2 SPARQL and named graph support

HyLAR+ relies on SPARQL to allow updating and querying the KB all along the application lifecycle. While the previous HyLAR version only supported SELECT queries, our framework now additionally supports INSERT DATA and DELETE DATA queries. It also handles named graphs for these three types of queries. Query answering is performed as follows. The reasoner receives as input a SPARQL query parsed into facts (triples) and an already classified graph representing an ontology or a named graph.

It applies OWL2-RL rules according to the chosen algorithm and iterates until no more facts can be inferred. The same operation is done with domain-specific rules, as processing inferences from the application domain must be done after deducing all OWL consequences of updated facts. Depending on the statement, it converts newly inferred implicit

[2]http://reasoning.restdesc.org/
[3]https://nodejs.org/
[4]https://code.google.com/p/owlreasoner/
[5]https://angularjs.org/
[6]Repo: https://github.com/ucbl/HyLAR/tree/IR-NG
Online demo: http://dataconf.liris.cnrs.fr/hylar
[7]https://code.google.com/p/trimpath/

	Causes	Consequences
scm-cso	T(?c1, rdfs:subClassOf, ?c2) T(?c2, rdfs:subClassOf, ?c3)	T(?c1, rdfs:subClassOf, ?c3)
scm-cls	T(?c, rdf:type, owl:Class)	T(?c, rdfs:subClassOf, ?c) T(?c, owl:equivalentClass, ?c) T(?c, rdfs:subClassOf, owl:Thing) T(owl:Nothing, rdfs:subClassOf, ?c)
cax-sco	T(?c1, rdfs:subClassOf, ?c2) T(?x, rdf:type, ?c1)	T(?x, rdf:type, ?c2)
cax-eqc1	T(?c1, owl:equivalentClass, ?c2) T(?x, rdf:type, ?c1)	T(?x, rdf:type, ?c2)
cax-eqc2	T(?c1, owl:equivalentClass, ?c2) T(?x, rdf:type, ?c2)	T(?x, rdf:type, ?c1)
eq-trans	T(?x, owl:sameAs, ?y) T(?y, owl:sameAs, ?z)	T(?x, owl:sameAs, ?z)
eq-syn	T(?x, owl:sameAs, ?y)	T(?y, owl:sameAs, ?x)

Figure 2: Supported OWL 2 RL semantics.

facts into an SQL query in order to update or select data from the KB.

4. EVALUATION

The location-agnostic feature of the HyLAR framework has already been evaluated in [8]. The reasoners embedded in the previous version do not follow the same OWL profile. Hence, as stated in Section 2.1, they do not cope with the same reasoning tasks and cannot be compared. In this section, we evaluate the performance of HyLAR+ w.r.t. two different aspects. First, we compare the execution times of the IR and greedy algorithms for ontology classification, as well as for INSERT DATA, DELETE DATA and SELECT statements. Second, we measure the impact of named graphs on these two algorithms while performing different SPARQL queries, in comparison to querying the whole KB.

In both evaluations, we use the FIPA-Device ontology[8] to calculate the classification times for both reasoning algorithms, once the ontology file has been parsed into a set of axioms and triples. This ontology comprises 14 classes, 14 object properties and 18 datatype properties as well a 67 axioms to express subsumptions, domains and ranges. We ran our evaluation on a Dell Inspiron, i7-2670QM CPU @ 2.20GHz. Computation times are in milliseconds.

4.1 Incremental reasoning evaluation

As IR reduces computation using ruleset restrictions, we chose the number of RL rules (see 3.1) as varying parameter. During this evaluation, we progressively added the following set of rules and applied them on FIPA:

R0 $= \emptyset$ R1 $=$ {`scm-cso`}
R2 $=$ R1 $\cup$ {`cax-sco`} R3 $=$ R2 $\cup$ {`scm-cls`}
R4 $=$ R3 $\cup$ {`cax-eqc1`} R5 $=$ R4 $\cup$ {`cax-eqc2`}
R6 $=$ R5 $\cup$ {`eq-trans`} R7 $=$ R6 $\cup$ {`eq-syn`}

[8]http://www.fipa.org/specs/fipa00091/PC00091A.html

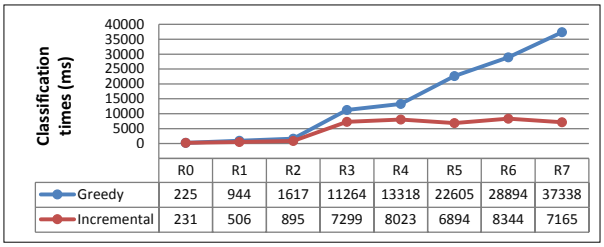

Figure 3: Classification times comparison for IR and greedy algorithm over a varied set of rules.

In Figure 3 we notice a gap between R2 and R3 which is due to our implementation choice: rules are currently represented with only one consequence. Thus, `scm-cls` splits into four sub-rules corresponding to its four consequences. Starting at R3, the IR classification time stabilizes, while the greedy algorithm increases as new rules are added to the set. We can assume that R3 is the point at which no more information in FIPA-Device is inferable from others rules. IR is more suitable in this situation as it does not compute over unnecessary rules.

4.2 Reasoning over named graphs

In order to measure the influence of named graphs on computation time, we evaluated the processing times for insertion, deletion and selection over the FIPA-Device ontology and subsumption rules from R2 (`scm-sco` and `cax-sco`). We consider four cases: IR (with no graph), IR (G) (on a named graph), GR (greedy reasoning with no graph) and GR (G) (greedy reasoning on a named graph). We evaluated these cases on 10, 20, 30, 40 and 50 triples. These data sets are available online, together with the framework.

The same behavior is observed on insertion and deletion. Without graphs, the query processing times increase faster as more triples are added/deleted, in comparison to updates using named graphs. At a certain point (50 triples when inserting and 20 triples when deleting), the greedy algorithm using named graphs is still better than the incremental without any graph. As expected, the best results are given by the IR with named graphs. For SELECT queries, processing times are totally stable: 8ms for any range of triples, including the entire KB. This is due to the fact that all implicit knowledge has already been inferred during the insertion, so the selection operation is straightforward.

5. DEMONSTRATION

During the demo, we will explain how HyLAR+ can be used to ease reasoning for a client-based Web application. We will demonstrate how our framework can locate a reasoning task on the client if possible or on the server otherwise. To highlight this, we developed a GUI in which the user can choose the reasoning algorithm and whether to use named graphs or not. The location of each reasoning task can be set manually or automatically. When set to automatic, the client checks its battery status and network connection (server ping) to decide where to locate the task. In a state-of-the-art WoT scenario on temperature regulation, we will load an ontology, classify it, and send several

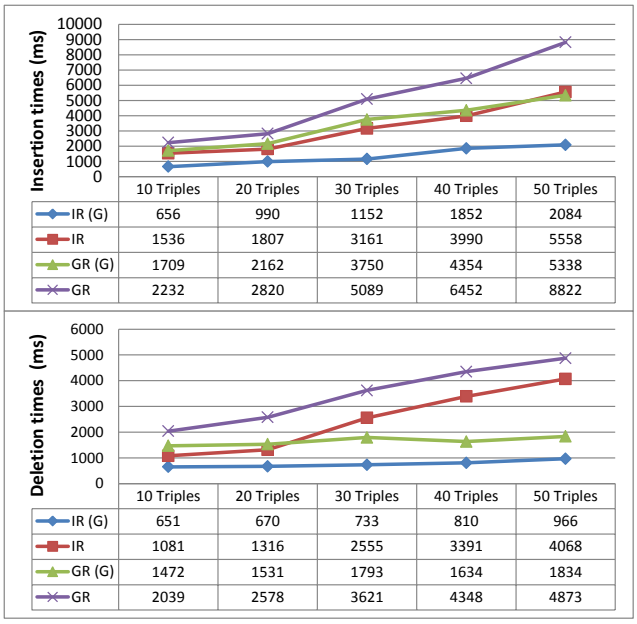

Figure 4: Evaluation of INSERT DATA and DELETE DATA queries with incremental and greedy reasoning, with and without named graphs.

SPARQL queries for insertion/deletion (update) and selection of instances, and compare processing times in different situations.

6. CONCLUSION

In this paper we present HyLAR+, a comprehensive reasoning framework that can perform reasoning tasks on both Web servers and users' clients. HyLAR+ is an improvement of a previous version and is designed to target Web applications in which some parts of data collection and processing are performed on the client side. Such applications usually handle small datasets and need to be reactive to data update and retrieval requests. To do so, HyLAR+ embeds a custom JavaScript rule-based reasoner that implements an incremental reasoning algorithm. HyLAR+ also supports named graphs and ships with a built-in set of OWL2-RL constructs. This rule set can be extended using first-order logic rules, either to improve the OWL2-RL specification coverage or to add domain-specific behaviors to an application. We present two evaluations that measure the reasoner efficiency regarding incremental reasoning and named graphs, for classification and update reasoning tasks.

During the demo, we will show the behavior of our reasoner with different settings and illustrate how to use HyLAR+ for importing ontologies from files and sending SPARQL INSERT DATA, DELETE DATA and SELECT queries. We will also show an example of adaptation decision heuristics to decide whether a query should be performed on the server or the client side.

The perspectives of our work are the following. We first want to make our framework compatible with the newly issued CHRjs[9] rule-based constraint solver, in order to compare our reasoner to a CHR-based one. Regarding the reasoner efficiency, we are working on still improving incremental reasoning performances by avoiding its over-deletion and rederivation steps. Regarding the decision of locating a reasoning task, we intend to provide a wider set of out-of-the box heuristics, a simple API to access them, as well as a set of benchmarks to help making decisions w.r.t. ontology size and number of rules. Using HyLAR+ and such extensions, Web developers will be easily able to leverage the power of semantic reasoning in small client-side applications.

[9]http://chrjs.net/

Acknowledgement

This work is supported by the French ANR (Agence Nationale de la Recherche) under the grant number <ANR-13-INFR-012>.

7. REFERENCES

[1] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus. Anapsid: An adaptive query processing engine for sparql endpoints. In *The Semantic Web–ISWC 2011*, pages 18–34. Springer, 2011.

[2] D. Estrin. Small data, where n= me. *Communications of the ACM*, 57(4):32–34, 2014.

[3] D. Guinard and V. Trifa. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, page 15, 2009.

[4] A. Gupta, I. S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. *ACM SIGMOD Record*, 22(2):157–166, 1993.

[5] I. Horrocks and P. Patel-Schneider. Knowledge representation and reasoning on the semantic web: OWL, 2010.

[6] M. Krötzsch. *OWL 2 Profiles: An introduction to lightweight ontology languages*. Springer, 2012.

[7] B. Motik, B. C. Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. Owl 2 web ontology language: Profiles. *W3C recommendation*, 27:61, 2009.

[8] M. Terdjimi, L. Médini, and M. Mrissa. Hylar: Hybrid location-agnostic reasoning. In *ESWC Developers Workshop 2015*, page 1, 2015.

[9] R. Verborgh, O. Hartig, B. De Meester, G. Haesendonck, L. De Vocht, M. Vander Sande, R. Cyganiak, P. Colpaert, E. Mannens, and R. Van de Walle. Querying datasets on the web with high availability. In *The Semantic Web–ISWC 2014*, pages 180–196. Springer, 2014.