A Reasoning Broker Framework for Protégé

Jürgen Bock, Tuvshintur Tserendorj,
Yongchun Xu, Jens Wissmann and Stephan Grimm

FZI Research Center for Information Technology
at the University of Karlsruhe
{bock,tserendo,xu,wissmann,grimm}@fzi.de

Abstract

We develop a reasoning broker framework for the combined use of existing OWL reasoners assisted by means for caching of results, scheduling of reasoning tasks and online-selection of appropriate reasoners. Here we present an integration of the first version of this reasoning broker framework into the Protégé ontology engineering environment. This allows for a controlled use of various reasoners within Protégé, supporting configurable strategies for selection and parallelisation.

1 Motivation

For the Web Ontology Language OWL, various systems for reasoning are available and being currently developed. Different systems focus on different aspects of OWL reasoning and are optimised for different types of reasoning, such as ABox or TBox reasoning. Moreover, there are specific research prototypes of reasoners designed for restricted OWL language fragments to achieve efficient reasoning by trading language expressivity for speed. Hence, not every reasoner can equally efficient process all kinds of reasoning tasks for every ontology. Some but not all of the available OWL reasoners can be readily used from within Protégé 4 by integration via the OWL API[1] [2], however, no combined use of reasoners is currently supported—a once selected reasoner is used for performing any reasoning within Protégé throughout.

This paper presents a reasoning broker system called HERAKLES[2] and its integration with Protégé 4. HERAKLES is designed to connect to different reasoners while it is itself acting as a single OWL reasoner. It redirects reasoning requests to the different reasoners in a way, such that the user can benefit from the strengths of all underlying external reasoners and their combination.

Protégé users who want to use a reasoner while authoring ontologies can now select HERAKLES as a reasoner via the user interface, and get a larger variety of underlying reasoners which are invoked intelligently and strategically regarding multiple reasoning requests, in order to deliver results more efficiently. Reasoning tasks can be processed in parallel by multiple reasoners and assigned to reasoners most suitable according to the type of reasoning requested, the parameters of the request or the characteristics of the ontology at hand.

In the following, we will sketch the features and design principles of our reasoning broker framework and reflect on its integration with Protégé 4.

2 Reasoning Brokerage

Our reasoning broker framework HERAKLES implements the reasoner interface provided by the OWL API and can thus be used as an ordinary OWL reasoner by any semantic application. However, behind the scenes HERAKLES manages several *external remote reasoners*, which can again be any reasoner that is accessible via the OWL API's reasoner interface. Currently there are connections to the standard reasoners Pellet[3], FaCT++[4], and HermiT[5], as well as to KAON2[6] via a newly developed adapter bridging

[1]`http://owlapi.sourceforge.net/`

[2]The presented research was funded by the German Federal Ministry of Economics (BMWi) under the project THESEUS (number 01MQ07019).

[3]`http://clarkparsia.com/pellet`

[4]`http://owl.man.ac.uk/factplusplus/`

[5]`http://www.hermit-reasoner.com/`

[6]`http://kaon2.semanticweb.org/`

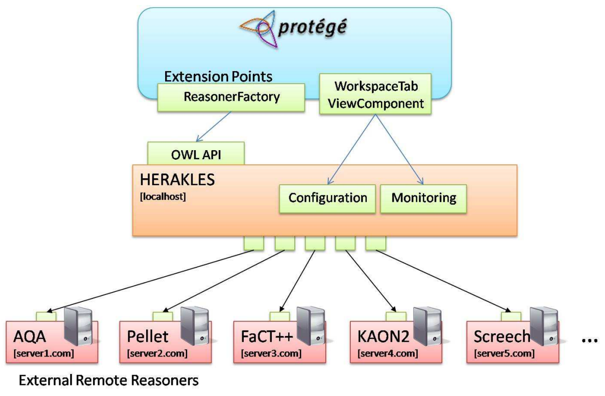


Figure 1: Overview of the HERAKLES plug-in

the KAON2 API and the OWL API. Furthermore we provide a connection to the approximate reasoning systems Screech [6] and AQA [5], which are both based on KAON2.

Figure 1 illustrates the general architecture of HERAKLES and its integration with Protégé. Apart from using HERAKLES as a reasoner via the OWL API, it can also be monitored and configured via tabs and views added to Protégé. The external remote reasoners are connected to HERAKLES via a remote interface, and can hence be located on different servers.

The centralised control of various external remote reasoners enables the implementation and combination of different features that provide added value compared to traditional reasoner systems. The following list of features can be implemented by the use of a strategy concept, which control the behaviour of the reasoning broker system. However, not all of those features have been implemented yet. Concrete strategies that are in fact implemented are mentioned at the end of this section.

Reasoner Selection Depending on the reasoning task to be performed, the most suitable reasoners among the available ones are automatically selected at runtime, taking into account the different specifics and strengths of a reasoner. Selection can depend on a combination of the type of reasoning task (*e.g.* ABox vs. TBox, parameters of a query, *etc.*), the properties of the ontology (*e.g.* expressivity, TBox/ABox ratio, *etc.*) and the characteristics of the reasoners.

Parallel Execution of Reasoning Tasks A reasoning task can be executed in parallel given a setting in which several reasoners are available, possibly running on remote machines. The first reasoner to finish can then propagate the result to Protégé, which results in a gain of efficiency. This way of parallelisation is currently implemented, however, we plan to extend this idea to splitting the query and executing subtasks in parallel. The effect of applying parallelisation also brings a benefit if several requests are to be answerd simultaneously or at frequent intervals. In that case, the quickest reasoner can immediately handle the next query, and as soon as other reasoners finish the initial task, they are available for consequent tasks.

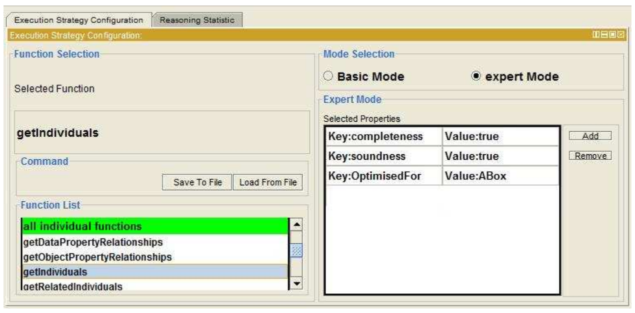
Partitioning of Ontologies By application of module extraction techniques such as described in [3], an ontology can be split in multiple partitions, such that for some reasoning tasks an execution on this partition might be sufficient, which also results in a gain of efficiency by reducing the size of the ontology to reason with. Query reformulation can then be accomplished similar to recent work by Clark&Parsia[7].

Load Balancing Having a multitude of external remote reasoners in the background, multiple reasoning requests can be handled concurrently by scheduling different tasks for different reasoners. The use of parallelisation and selection also allows for answering a larger number of consecutive requests.

Real-time Benchmarking Parallel execution of different external remote reasoners allows for constantly benchmarking them during execution. These statistics can then be used for future reasoner selections, in order to predict the most adequate reasoners with higher confidence.

Utilisation of the different external remote reasoners is managed by two kinds of strategies: a *load strategy* to manage the loading of ontologies into different external remote reasoners, and an *execution*

[7]`http://clarkparsia.com/weblog/2009/02/04/distributed-query-pellet-into-the-void/`



Selected Function: getIndividuals

Mode Selection: Basic Mode [✓ expert Mode]

Selected Properties

Key:completeness	Value:true
Key:soundness	Value:true
Key:OptimisedFor	Value:ABox

Function List:
- all individual functions
- getDataPropertyRelationships
- getObjectPropertyRelationships
- getIndividuals
- getRelatedIndividuals

Figure 2: Part of the HERAKLES configuration tab.

strategy to delegate API methods for reasoning tasks. Load and execution strategy are both exchangeable, such that the behaviour of the reasoning broker can be controlled by using customised strategies.

In the current HERAKLES release we provide several strategies, which realise some, though not yet all of the features mentioned above.

A *BasicLoadStrategy* loads a set of ontologies into all available reasoners. A range of execution strategies are available. The *BasicParallelisationStrategy* executes a reasoning request on all available reasoners in parallel and returns the result of the reasoner, which finishes first. An *ABoxTBoxRatioSelectionStrategy* selects a set of reasoners which are supposed to deal best with ontologies that have a certain ratio of ABox and TBox sizes. This information is part of the properties of each reasoner and is based on benchmarking results [1]. A *TaskSelectionStrategy* selects suitable reasoners according to the information about which reasoners work best for which reasoning tasks. This strategy can be configured in detail from within Protégé, as will be discussed in Sect. 3. A strategy independent caching mechanism allows for the efficient handling of frequent reasoner invocations, *e.g.* by the Protégé UI.

In addition to the OWL API reasoner interface, we provide an interface for anytime reasoning, which is also implemented by HERAKLES. This allows for an asynchronous communication with the reasoner, where available results are noticed by a listener continuously without the need to wait for the reasoner to finish. Currently, an *AnytimeStrategy* is realised by using a combination of different approximate reasoners and their properties of delivering results that are guaranteed to be sound but possibly incomplete or vice versa.

3 Integration into Protégé

We developed a plug-in[8] for Protégé 4. Protégé provides several OSGi extension points to include additional functionality. We integrated the HERAKLES reasoning broker using the *ReasonerFactory* extension point, which makes it selectable from within Protégé, and hence can be used like any other OWL API reasoner. We also provide user interface components to configure the broker in terms of selection of remote reasoners, as well as selection and configuration of broker strategies. This is provided by a new *configuration tab*. Additionally the usage of the external remote reasoners can be monitored. Configurations can be persisted in configuration files for reuse.

Selection of External Remote Reasoners The configuration tab displays all connected remote reasoners, and allows to add or remove additional reasoners that can then be used by the strategies. The user can see the reasoner properties, such as name of reasoner (Pellet, FaCT++, *etc.*), supported language fragment, soundness/completeness characteristics, *etc.*, which helps to decide whether a particular reasoner is to be included or not. In a specific scenario, for instance, only reasoners that deliver correct results might be of interest, while approximate reasoning systems are to be disregarded.

Strategy Configuration The configuration tab also allows for the selection and configuration of load and execution strategies to be used by HERAKLES. Strategy configuration is currently realised for the *TaskSelectionStrategy*, described in Sect. 2 that allows to assign any reasoner function to a reasoner identified by a set of properties. The selectable tasks correspond to OWL API methods, such as *getIndividuals*, or *getSubClasses*.

The configuration tab allows to define the conditions under which a reasoner can be selected to execute these functions, based on a combination of reasoner properties. The user can choose between

[8]`http://www.fzi.de/downloads/ipe/herakles/herakles.zip`

two different modes to configure the conditions. In *basic mode* a user just selects the reasoner type, which should be used for a specific API method. In *expert mode* selection conditions can be defined manually using any properties a reasoner needs to match. The property constraints can be defined via conjunctive or disjunctive combination of conditions. For instance, a user can define the condition, that for an instance retrieval query via the *getIndividuals* method only sound and complete reasoners are to be selected, which are optimised for ABox reasoning. This scenario is illustrated in Fig. 2.

Monitoring of External Remote Reasoners We also provide a monitoring component to display information about external remote reasoners and to show the status of currently running tasks. As the reasoners are run competitively, the number of times a reasoner responded first to a given task is displayed along with the percentage relative to the other reasoners. This information can be useful for developers of reasoners, as well as for the refinement of the HERAKLES configuration. For instance, if a certain reasoner performs best in a particular scenario, more instantiations of this reasoner can be set up and connected to HERAKLES, in order to further increase the overall performance.

Anytime Querying As HERAKLES supports anytime behaviour through the asynchronous anytime reasoner interface, we also provide an *anytime reasoning tab* for the successive retrieval of query results. It extends the Protégé 4 DL Query tab by the feature of timely decoupled displaying of query results that are incrementally expanded or refined in an asynchronous manner, which allows to present early results before the underlying reasoning task is completed. In combination with the approximate reasoning systems mentioned before, we further exploit reasoner properties of soundness and completeness to qualify query results by means of different colours and strike-through rendering[9]. In one particular scenario, the use of an unsound but complete reasoner provides early results for instance retrieval queries, which are then either confirmed or revised subsequently by a sound but incomplete counter-part run in parallel. (See [4] for a description of such scenarios.)

4 Outlook

The next steps in the development of HERAKLES include the realisation of the features listed in Sect. 2 that are not yet addressed by existing strategies. More concretely we plan to incorporate modularisation/partitioning for ontologies and queries, as well as more sophisticated reasoner selection, and load balancing.

References

[1] Jürgen Bock, Peter Haase, Qiu Ji, and Raphael Volz. Benchmarking OWL Reasoners. In *Proceedings of the ARea2008 Workshop*, Tenerife, Spain, June 2008.

[2] Matthew Horridge, Sean Bechhofer, and Olaf Noppens. Igniting the OWL 1.1 Touch Paper: The OWL API. In Christine Golbreich, Aditya Kalyanpur, and Bijan Parsia, editors, *OWLED*, volume 258 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[3] Ernesto Jimenez-Ruiz, Bernardo Cuenca Grau, Ulrike Sattler, Thomas Schneider, and Rafael Berlanga. Safe and economic re-use of ontologies: A logic-based methodology and tool support. In *OWL: Experiences and Directions (OWLED)*, 2008.

[4] Sebastian Rudolph, Tuvshintur Tserendorj, and Pascal Hitzler. What Is Approximate Reasoning? In *RR'08: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems*, pages 150–164, Berlin, Heidelberg, 2008. Springer-Verlag.

[5] Tuvshintur Tserendorj, Stephan Grimm, and Pascal Hitzler. Approximate Instance Retrieval. Technical report, FZI at University of Karlsruhe, Germany, December 2008. availavle at `http://www.aifb.uni-karlsruhe.de/WBS/phi/resources/publications/approxInstRetr08.pdf`.

[6] Tuvshintur Tserendorj, Sebastian Rudolph, Markus Krötzsch, and Pascal Hitzler. Approximate OWL-Reasoning with Screech. In *RR'08: Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems*, pages 165–180, Berlin, Heidelberg, 2008. Springer-Verlag.

[9] Results from approximate reasoning systems are possibly incorrect or incomplete, which makes it necessary to confirm preliminary results by using different colours, or to invalidate them by striking them through.