Towards the Evaluation of the LarKC Reasoner Plug-ins

Zhisheng Huang
huang@cs.vu.nl

Computer Science Department, Vrije Universiteit Amsterdam, The Netherlands

Abstract. In this paper, we present an initial framework of evaluation and benchmarking of reasoners deployed within the LarKC platform, a platform for massive distributed incomplete reasoning that will remove the scalability barriers of currently existing reasoning systems for the Semantic Web. We discuss the evaluation methods, measures, benchmarks, and performance targets for the plug-ins to be developed for approximate reasoning with interleaved reasoning and selection. In this paper, we propose a specification language of gold standards for the evaluation and benchmarking, and discuss how it can be used for the evaluation of reasoner plug-ins within the LarKC platform.
keywords: Reasoning, Evaluation, Benchmarking, Web Scale Reasoning

1 Introduction

The essence of the LarKC project[2][1] is to go beyond notions of absolute correctness and completeness in reasoning. We are looking for retrieval methods that provide useful responses at a feasible cost of information acquisition and processing. Therefore, generic inference methods need to be extended to non-standard approaches. In consequence, traditional metrics such as completeness and correctness for reasoning need to be replaced by metrics that ratio the utility of solutions with their related costs as a means to evaluate the chosen problem solver. Therefore, we will develop a framework for evaluation and measuring the relative utility of various reasoning approaches that will be implemented in the LarKC project.

Approximate reasoning is non-standard reasoning which is based on the idea of sacrificing soundness or completeness for a significant speed-up in reasoning. This is to be done in such a way that the loss of correctness is at least outweighed by the obtained speed-up [8]. Anytime reasoning in which more answers can be obtained over time is expected to be a behavior of approximate reasoning for the LarKC platform. Interleaving reasoning and selection is considered to be an approach to improving the performance of the LarKC platform[7, 4]. The main idea of the interleaving framework is to use selectors to select only limited and relevant part of data for reasoning, so that the efficiency and the scalability of reasoning can be improved. Those non-standard reasoning approaches need

[1] http://www.larkc.eu

new metrics and frameworks for the evaluation and benchmarking of reasoners developed within the LarKC platform.

These new reasoning paradigms, which fuse approaches from many different fields will also require new approaches to evaluation. Traditional measures like soundness and completeness will have to be enriched with measures such as recall and precision, and worst-case complexity will have to be enriched by approaches such as anytime performance profiles. In this paper, we will develop such new evaluation measures and apply them to the implemented resoner plug-ins, both on synthetic datasets and on datasets from the use-cases.

In this paper, we will present an initial framework of evaluation and benchmarking of reasoners deployed with the LarKC platform. We will discuss the evaluation methods, measures, benchmarks, and performance targets for the plug-ins to be developed for the task of approximate reasoning with interleaved reasoning and selection. Furthermore, we will develop a specification language of gold standards for the evaluation and benchmarking.

The rest of the paper is organized as follows. In Section 2, we overview the LarKC Platform and the general picture of reasoner plug-ins, which will be developed or deployed within the LarKC platform. In Section 3, we develop a framework of evaluation and benchmarking of reasoners. In Section 4, we propose a specification language of gold standards for the evaluation and benchmarking within the LarKC platform. Section 5 discusses the related work before concluding the paper in Section 6.

2 LarKC Platform

2.1 LarKC Architecture

In [9], the first version of the LarKC architecture has been proposed. This design is based on a thorough analysis of the requirements and considering the lessons learned during the first year of the project. Figure 1 shows a detailed view of the LarKC Platform architecture.

The LarKC platform has been designed in a way so that it is as lightweight as possible, but provides all necessary features to support both users and plug-ins. For this purpose, the following components are distinguished as part of the LarKC platform:

- **Plug-in API:** defines interfaces for plug-ins and therefore provides support for interoperability between platform and plug-ins and between plug-ins.
- **Data Layer API:** provides support for data access and management.
- **Plug-in Registry:** contains all necessary features for plug-in registration and discovery
- **Workflow Support System:** provides support for plug-in instantiation, through the deployment of plug-in managers, and for monitoring and controlling plug-in execution at workflow level.

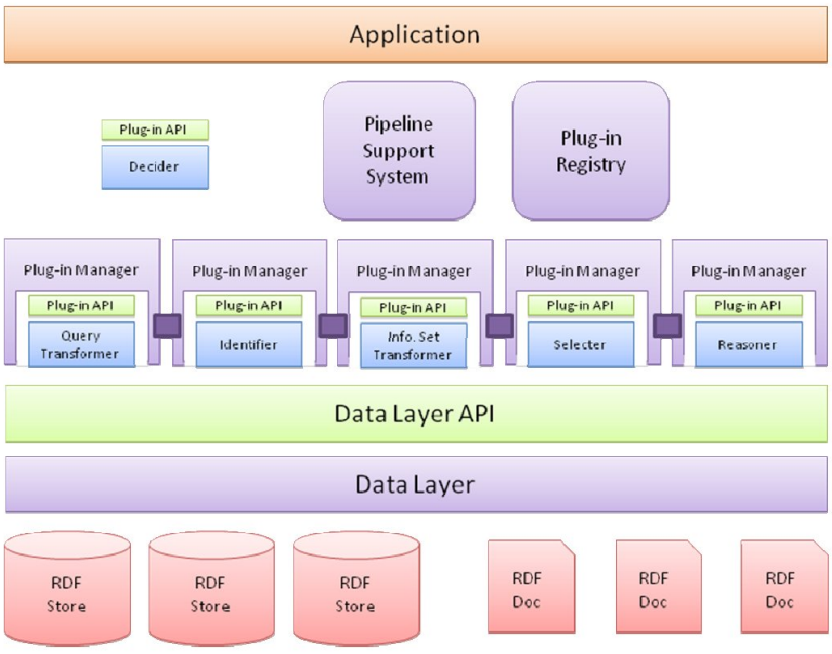

Fig. 1. The LarKC Platform Architecture

- **Plug-in Managers:** provides support for monitoring and controlling plug-ins execution, at plugin level. An independent instance of a Plug-in Manager is deployed for each plug-in to be executed. This component includes the support for both local and remote execution and management of plug-ins.
- **Queues:** provides support for deployment and management of the communication between platform and plug-ins and between plug-ins.

2.2 LarKC resoner plug-ins

Reasoning APIs All LarKC plug-ins share a common super class, namely the Plug-in class. This class provides an interface for functions common to all plug-in types. All plug-ins are identified by a name, which is a string. Plug-ins provide meta data that describes the functionality that they offer. Plug-ins provide Quality of Service (QoS) information regarding how they perform the functionality that they offer.

The resoner plug-in will execute a given SPARQL Query against a Triple Set provided by a Select plug-in. The interface of the resoner plug-in can be seen in Table 1.

The resoner plug-in supports the four standard methods for a SPARQL endpoint, namely select, describe, construct, and ask. The input to each of the reason methods are the same and includes the query to be executed, the statement set

Table 1. Reasoner Plug-in Interface

Function name
sparqlSelect(SPARQLQuery q, SetOfStatements s, Contract c, Context ctx)
sparqlConstruct(SPARQLQuery q, SetOfStatements s, Contract c, Context ctx)
sparqlDescribe(SPARQLQuery q, SetOfStatements s, Contract c, Context ctx)
sparqlAsk(SPARQLQuery q, SetOfStatements s, Contract c, Context ctx)

to reason over, the contract, which defines the behavior of the reasoner, and the context, which defines the special information of the reasoning task. The output of these reasoning methods depends on the reasoning task being performed. The select method returns a Variable Binding as output where the variables correspond to those specified in the query. The construct and describe methods return RDF graphs, in the first case this graph is constructed according to the query and in the second the graph contains triples that describe the variable specified in the query. Finally ask returns a Boolean Information Set as output, which is true if the pattern in the query can be found in the Triple Set, or false if not.

Reasoner Plug-ins The LarKC reasoner plug-ins can range from the reasoners which provide the standard reasoning support with RDF/RDFS/OWL data to the reasoners which realize non-standard reasoning tasks such as reasoning with inconsistent ontologies, rule-based reasoning, stream reasoning. Here is an (incomplete) list of the LarKC resoner plug-ins which have been developed for the LarKC platform.

- **SPARQL Query Evaluation Reasoner**: This resoner plug-in wraps OWLIM and enables the execution of SPARQL Select, Construct, Describe and Ask queries to be executed against it.
- **Pellet Reasoner**: This resoner plug-in is a wrapper of Pellet SPARQL DL Reasoner[2], which provides the reason support of Description Logics.
- **DIG Interface**: This resoner plug-in provides the support for the DIG interface[3], which allows an external DIG reasoner to be called, like RACER, FACT++,KAON2, PION, etc.
- **OWLAPI Reasoner**: This resoner plug-in provides the support for OWL APIs, which is a standard for reasoners with OWL data.
- **PION Reasoner**: This is a reasoner which can be used for reasoning with inconsistent ontologies. Namely, given an inconsistent ontology and a query, the PION reasoner can return a meaningful answer.

[2] http://clarkparsia.com/pellet
[3] http://dig.sourceforge.net/

3 A Framework of Evaluation and Benchmarking for Ontology Reasoners

3.1 General Consideration

In this section, we will present a framework of evaluation and benchmarking for ontology reasoners. The main idea is to use the framework, which have developed in the KnowledgeWeb project for benchmarking inconsistency reasoners in the Semantic Web[6]. In LarKC, we can use this framework to evaluate and benchmark both standard reasoner plug-ins and non-standard reasoner plug-ins, e.g. the PION reasoner[5, 3] for reasoning with inconsistent ontologies.

In ontology engineering, evaluation and benchmarking target software products, tools, services, and processes. The objects are called tested systems. Evaluation and benchmarking are the systematic determination of merit, worth, and significance of tested systems. Those merit, worth, and significance are characterized as a value relation, which is considered as a preference relation, i.e., a partial order set $\langle A, \succ \rangle$. We consider a tested system as one which is targeted by the objectives of evaluation or benchmarking. A tested system can be characterized as an input-output function, alternatively, called a characteristic function of the tested system. Namely, it maps a tuple of the input parameters into an output value. A value relation is defined as a preference relation on a set of values. Namely, a value relation is characterized as a partial order set.

We define *evaluation* as the systematic determination of the values of tested systems with respect to its partial ordered value relation, whereas *benchmarking* as a continuous process for improving by systematically evaluating tested systems, and comparing them to those considered to be the best. Namely, benchmarking is the continuous process of evaluation.

In the LarKC project, Task 4.7.1 is requested to develop an initial framework for evaluation of reasoner plug-ins. For the purpose of continuous improvement of the LarKC platform, the issue of benchmarking reasoner plug-ins should be also covered.

3.2 Goals and Criteria for Evaluation and Benchmarking of Reasoner Plug-ins

We consider the following initial goals for evaluating LarKC Reasoner plug-ins:

- **Bug Detection:** A good evaluation of reasoner plug-ins should be able to detect hidden bugs in the implementation. These bugs may be hard to detect with manual examination by developers. It requires that test data sets cover many functionalities/use cases of reasoner plug-ins.
- **Robustness:** A robust reasoner plug-in should not fail with noisy or erroneous test data. Thus, special test data sets should be designed to test the robustness of a reasoner plug-in.
- **Performance analysis:** One of the main concerns on the quality of a reasoner plug-in is its performance. Thus, a necessary procedure of evaluation

and benchmarking of reasoner plug-ins is to provide an analysis of their performance. The usual criteria for examining the performance of reasoner plug-ins are: (i) the time costs, including the time cost for getting the first query answer with anytime behavior, and the average time cost for each query answer, (ii) the resource consumption, including the maximal working memory request, and (iii) the quality of the query answers, which will be discussed in the next section.
- **Scalability Potential:** The LarKC platform is expected to support Web scale reasoning. Thus, the scalability of a reasoner plug-in becomes a crucial issue for the performance of the overall platform. The scalabililty potential of a reasoner is how well it can deal with large amount of data.
- **Platform Improvement:** A useful evaluation and benchmarking of reasoner plug-ins should be able to find bottle necks within the platform. It would provide an analysis of how the design of platform can be improved.

3.3 Measuring the Quality of Query Answers

As discussed in the last section, the quality of query answers is one of the main criteria for evaluating and benchmarking of reasoners.

The answer value set for standard ontology reasoning is usually considered as a Boolean value set, namely, it consists of 'true' and 'false'. The answer value set for reasoning with inconsistent ontologies usually consists of three values $accepted, rejected$, and $undetermined$, as introduced in the PION system. We will develop gold standards, which represents intuitive answers from a human for queries on reasoning with consistent or inconsistent ontologies. Thus, we can compare the answers from the tested system/approach with the gold standard, which is supposed to be intuitively true by a human to see to what quality of query answers provided by tested systems.

For a query with an inconsistent ontology, there might exist the following difference between an answer from the tested system/approach and its intuitive answer in a gold standard.

- **Intended Answer**: the system's answer is the same as the intuitive answer;
- **Counter-intuitive Answer**: the system's answer is opposite to the intuitive answer. Namely, the intuitive answer is 'accepted' whereas the system's answer is 'rejected', or vice versa.
- **Cautious Answer**: The intuitive answer is 'accepted' or 'rejected', but the system's answer is 'undetermined'.
- **Reckless Answer**: The system's answer is 'accepted' or 'rejected' whereas the intuitive answer is 'undetermined'. We call it a reckless answer, because under this situation the system returns just one of the possible answers without seeking other possibly opposite answers, which may lead to 'undetermined'.

Therefore, a value set

$$\{intended\_answer, cautious\_answer, reckless\_answer, counter\_intuitive\_answer\},$$

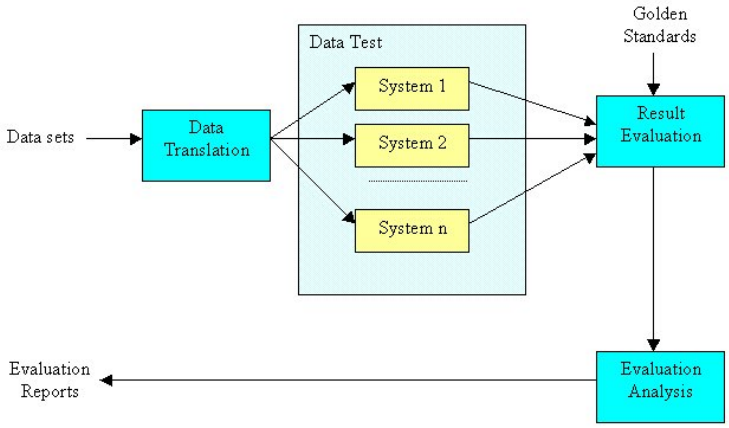

Fig. 2. Workflow of evaluation.

can be introduced for the evaluation of answers with gold standards. An intended answer is considered as a best one, whereas a counter intuitive answer is considered as a worse one. Cautious answers are usually not considered as wrong answers, whereas reckless answers may give wrong answers. Thus, a preference relation on the value set can be like this:

$\{intended\_answer \succ cautious\_answer,$
$cautious\_answer \succ reckless\_answer,$
$reckless\_answer \succ counter\_intuitive\_answer\}$

Based on this preference order, we can measure the quality of query answers by the following answer rates:

- **IA Rate**, which counts only intended answers. Namely the Intended Answer Rate is defined as the ratio of the amount of Intended Answers to the total amount of the answers.
- **IC Rate**, which counts non-error answers. Namely, IC Rate = (Intended Answers +Cautious Answers)/TotalAnswerNumber.

3.4 Workflows of Evaluation and Benchmarking

Common data sets and common gold standards are usually used for an evaluation of different tested systems. Those systems may be heterogeneous with respect to their input data. For example, a reasoner may support only OWL data, whereas another reasoner may support only DIG data. Therefore a data translator is needed to convert data sets represented in a standard format into the data sets which are represented in a format that is supported by a tested system. Based on a comparison between test results and gold standards, result evaluation can be done manually, semi-automatically, or automatically. The output of the result

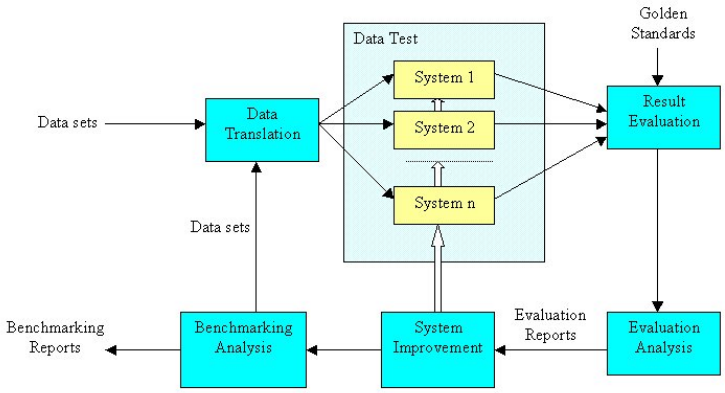

Fig. 3. Workflow of benchmarking

evaluation and the implication are further analyzed by an evaluation analysis. The methods of statistics and visualization are usually introduced in this phase for better illustration. The evaluation results will be ranked with respect to its value relation. Finally, it leads to an evaluation report which concludes the values of tested systems and explain the reasons why the system behaves differently. An investigation is usually made to detect the problem of tested systems based on the analysis of the evaluation. The workflow of evaluation is shown in Figure 2.

As discussed above, benchmarking is a continuous processing of evaluation. Therefore for benchmarking, evaluation results are used further for the improvement of tested systems. This would usually lead to new versions of tested systems. Based on a benchmarking analysis, new test data sets may be re-designed or previous data sets are adjusted for further evaluation with respect to some targeted problems. The workflow of benchmarking is shown in Figure 3.

4 A Specification Language for Gold Standards

Manual evaluation and analysis of test results are usually time consuming, labor intensive, and error prone. The formalism of gold standard will pave a way for automatic or semi-automatic evaluation and analysis of test results.

A gold standard is an evaluation function which maps queries into answers with confidence values. For reasoner benchmarking, a gold standard is a (partial) function which maps queries into (intuitive) answers with a confidence value. For example, for benchmarking inconsistency processing, we considered the answer set $\{accepted, rejected, undetermined\}$. For a query "are birds animals?", the expected answer is intuitively considered as "accepted" with confidence value "1.0". However, for the query "are men animals?", the expected answers may be well suitable to be specified as an answer with lower confidence value, say, "ac-

Table 2. Example of Gold Standard

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <goldenStandard xmlns="http://wasp.cs.vu.nl/larkc/d471/lang"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://wasp.cs.vu.nl/larkc/d471/gd.xsd">
  <name value="LarKC gold standard example 1" version="0.0.1"/>
  <comment text="just an example, which is independent from any
    particular ontology. It is up to evaluators to decide
    which ontology can be applied"/>
      <query id="Are birds animals?" querytype="subsumes">
        <queryBody>  ...  </queryBody>
        <expectedAnswers>
          <answer value="accepted" confidence="1"/>
        </expectedAnswers>
      </query>
      <query id="Are men animals?" querytype="subsumes">
        <queryBody>  ...  </queryBody>
        <expectedAnswers>
          <answer value="accepted" confidence="0.4"/>
          <answer value="undetermined" confidence="0.2"/>
          <answer value="rejected" confidence="0.4"/>
          <comment text="just an example which shows the possibility
          of multiple answers in a gold standard"/>
        </expectedAnswers>
     </query>
</goldenStandard>
```

cepted" with confidence value "0.4", "rejected" with confidence value "0.4", and "undetermined with the confidence value "0.2". Namely, we use the confidence values to represent some kinds of uncertainty of expected answers. The confidence values can be obtained by various approaches, like from questionnaires, statistics, machine learning, etc.

We design gold standards which are independent from a specific ontology. Namely, it is up to evaluators/users to decide which ontologies can be applied with respect to a gold standard.

In the following, we develop a gold standard specification language which is suitable for SPARQL queries as reasoning queries. Thus, it is an XML file, which is easy to use and read. Table 2 shows an example of a gold standard which is encoded as an XML document.

This XML document specifies the name and the version of the gold standard. Each query consists of a detailed query statement (in the SPARQL query language) and its expected answer specification. Each expected answer is attached by a confidence value. For non-Boolean answers, like those for sparqlSelect and sparqlConstruct queries which would return a variable binding or a rdf graph

Table 3. XML Format for Expected Answers

```
...
<query id="List all the subconcepts fo wine"
              querytype="sparqlSelect">
  <queryBody>
   <sparqlPrefix name="rdfs"
      value="http://www.w3.org/2000/01/rdfschema#/>
   <sparqlPrefix name="wine"
      value="http://www.w3.org/TR/2003/PR-owl-guide-20031209/wine#"/>
   <sparqlBody value="SELECT ?X WHERE {?X rdfs:subClassOf wine:Wine.}"/>
  </queryBody>
  <expectedAnswers>
   <answer confidence="1.0"/>
    <value>......</value>
  </expectedAnswers>
...
```

Table 4. RDF representation of the Gold standards

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:fields="http://sindice.com/vocab/fields#"
          ......
         xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:Description
    rdf:about="http://wasp.cs.vu.nl/larkc/d471/lang#GoldStandardName">
  <larkc:name>LarKC gold standard example 1</larkc:name>
  </rdf:Description>
   <rdf:Description
     rdf:about="http://wasp.cs.vu.nl/larkc/d471/lang#Query">
      <larkc:queryID>Are birds animals?</larkc:queryID>
      <larkc:queryType>subsumes</larkc:queryType>
      <larkc:queryBody>...</larkc:queryBody>
      <larkc:expectedAnswers>
        <rdf:Bag><rdf:li><larkc:answer>
            <larkc:value>accepted</larkc:value>
            <larkc:confidence>1</larkc:confidence></larkc:answer></rdf:li>
         </rdf:Bag>
        </larkc:expectedAnswers>
   </rdf:Description>
      ......
</rdf:RDF>
```

as an answer, the values of expected answers are specified with a detailed xml-encoded subtree as specified in Table 3.

Alternatively, the gold standard specification language can be defined by using the standard meta data languages or ontology language, such as RDF/RDFS/OWL. Namely, a gold standard can be specified as meta data or an ontology, which provides a possibility for reasoning with gold standards. Table 4 shows an example of the RDF representation of the gold standards.

5 Related Work

In [1], Castro develops an evaluation and benchmarking methodology for Semantic Web technologies and presents various methods for RDF(S) and OWL interoperatability benchmarking.

SEALS[4] is a project on Semantic Evaluation at Large Scale. The goal of the SEALS project is to provide an independent, open, scalable, extensible, and sustainable evaluation infrastructure for semantic technologies. The SEALS Platform allows the remote evaluation of semantic technologies thereby providing an objective comparison of the different existing semantic technologies. The SEALS Platform will provide an integrated set of semantic technology evaluation services and test suites. Therefore, one of the future work is to integrate the evaluation methods and benchmarks developed in the context of LarKC with the SEALS Platform.

The work on the evaluation design for advanced reasoning systems in the SEALS project[10] is still under development. We have observed that the SEALS project has presented the definition of the evaluations and test data that will be used in the first SEALS Evaluation Campaign. The tests have been designed to cover the interoperability and the performance of advance reasoning systems.

6 Conclusions

In this paper, we have presented an initial framework of evaluation and benchmarking for reasoner plug-ins within the LarKC platform. We have proposed the evaluation methods, measures, benchmarks, and performance targets for the plug-ins to be developed for approximate reasoning with interleaved reasoning and selection. Based on the initial framework, we have discussed the workflows of evaluation and benchmarking. Furthermore, in this paper, we have proposed a specification language of gold standards for evaluation and benchmarking. We have discussed how the proposed language of gold standards can be used for the evaluation of reasoner plug-ins within the LarKC platform.

References

1. R. G. Castro. *Benchmarking Semantic Web Technology.* IOS Press, 2010.

[4] http://www.seals-project.eu/

2. D. Fensel, F. van Harmelen, B. Andersson, P. Brennan, H. Cunningham, E. D. Valle, F. Fischer, Z. Huang, A. Kiryakov, T. K. Lee, L. School, V. Tresp, S. Wesner, M. Witbrock, and N. Zhong. Towards larkc: A platform for web-scale reasoning. In *Proceedings of the International Conference on Semantic Computing*, pages 524–529, 2008.
3. Z. Huang and F. van Harmelen. Using semantic distances for reasoning with inconsistent ontolgies. In *Proceedings of the 7th International Semantic Web Conference (ISWC2008)*, 2008.
4. Z. Huang, F. van Harmelen, S. Schlobach, G. Tagni, A. ten Teije, Y. Wang, Y. Zeng, and N. Zhong. D4.3.2 - implementation of plug-ins for interleaving reasoning and selection, March 2010. Available from: http://www.larkc.eu/deliverables/.
5. Z. Huang, F. van Harmelen, and A. ten Teije. Reasoning with inconsistent ontologies. In *Proceedings of the International Joint Conference on Artificial Intelligence - IJCAI'05*, 2005.
6. Z. Huang, J. Volker, Q. Ji, H. Stuckenschmidt, C. Meilicke, S. Schlobach, F. van Harmelen, and J. Lam. Benchmarking the processing of inconsistent ontologies, knowledgeweb d1.2.2.1.4/d2.1.6.3, 2007. Available from: http://wasp.cs.vu.nl/knowledgeweb/D2163/kweb2163.pdf.
7. Z. Huang, Y. Zeng, S. Schlobach, A. den Teije, F. van Harmelen, Y. Wang, and N. Zhong. D4.3.1 - strategies and design for interleaving reasoning and selection of axioms, September 2009. Available from: http://www.larkc.eu/deliverables/.
8. S. Rudolph, T. Tserendorj, and P. Hitzler. What is approximate reasoning? In *Proceedings of RR2008, LNCS 5341*, pages 150–164, 2008.
9. M. Witbrock, B. Fortuna, L. Bradesko, M. Kerrigan, B. Bishop, F. van Harmelen, A. ten Teije, E. Oren, V. Momtchev, A. Tenschert, A. Cheptsov, S. Roller, and G. Gallizo. Larkc deliverable d5.3.1 - requirements analysis and report on lessons learned during prototyping, June 2009. Available from: http://www.larkc.eu/deliverables/.
10. M. Yatskevich, G. Stoilos, I. Horrocks, and F. Martin-Recuerda. Seals deliverable d11.2, evaluation design and collection of test data for advanced reasoning systems, 2009.