Guidelines for Benchmarking the Performance of Ontology Management APIs

Raúl García-Castro and Asunción Gómez-Pérez

Ontology Engineering Group, Laboratorio de Inteligencia Artificial,
Facultad de Informática, Universidad Politécnica de Madrid, Spain
{rgarcia, asun}@fi.upm.es

Abstract. Ontology tools performance and scalability are critical to both the growth of the Semantic Web and the establishment of these tools in the industry. In this paper, we present briefly the benchmarking methodology used to improve the performance and the scalability of ontology development tools. We focus on the definition of the infrastructure for evaluating the performance of these tools' ontology management APIs in terms of its execution efficiency. We also present the results of applying the methodology for evaluating the API of the WebODE ontology engineering workbench.

1 Introduction

The lack of mechanisms to evaluate ontology tools is an obstacle to their use in companies. Performance is one of the critical requirements requested for ontology tools and the scalability of these tools is a primary need.

To the best of our knowledge, no one has evaluated ontology development tools according to their performance. Some general evaluation frameworks for ontology tools have been proposed by: Duineveld et al. [1], the deliverable 1.3 of the OntoWeb project [2] and Lambrix et al. [3]; and the EON workshops series [4, 5, 6] focus on the evaluation of ontology tools but they have not dealt with their performance yet.

The evaluation of the performance of ontology development tools is tightly related to the evaluation of their scalability. To this end, the tools must be evaluated according to different workloads, paying special attention to the effect of high workloads on the tool performance. Magkanaraki et al. [7] and Tempich and Volz [8] performed structural analyses of ontologies in order to define these workloads. Workload generators such as OntoGenerator [2] and the Univ-Bench Artificial data generator [9] produce ontologies for performing experiments in an automatic way and according to some parameters.

In this paper, we present an approach and a realization of a benchmarking methodology with regard to the performance and the scalability of ontology development tools. The advantage of using a benchmarking methodology rather than an evaluation one is that developers will be able to obtain both a continuous improvement in their tools and the best practices that are performed in the area, supporting the industrial applicability of ontology tools.

As we will see in the next section, experimentation is a key part of any benchmarking methodology. This paper presents a general infrastructure to evaluate

Y. Gil et al. (Eds.): ISWC 2005, LNCS 3729, pp. 277 – 292, 2005.

the performance and the scalability of ontology development tools by assessing the performance of the methods of their ontology management APIs in terms of their execution efficiency.

It also presents the results of applying the proposed infrastructure for evaluating the performance and the scalability of the ontology management API of the WebODE ontology engineering workbench. WebODE [10] provides services for editing and browsing ontologies, for importing and exporting ontologies to classical and semantic web languages, for evaluating ontologies, for mapping ontologies, etc. As we need a tool for generating ontologies in WebODE's knowledge model, we have developed the WebODE Workload Generator that generates synthetic WebODE ontologies according to a predefined structure and to a load factor.

The contents of this paper are the following: Section 2 presents the benchmarking methodology for ontology tools. According to this methodology, Section 3 presents the benchmarking goal and the metrics to be used for evaluating the performance of the ontology management APIs of ontology development tools; Section 4 presents a detailed definition of the infrastructure needed for evaluating the performance of these APIs and an explanation of how this infrastructure was instantiated for evaluating WebODE's API. Sections 5 and 6 present the evaluation of WebODE's API and the analysis of the results of this evaluation, respectively. Finally, Section 7 presents the conclusions obtained and the related future work.

Out of the scope of this paper are other evaluation criteria like stability, usability, interoperability, etc. as well as the evaluation of the performance of other ontology development tool functionalities such as user interfaces, reasoning capabilities when dealing with complex queries, or ontology validators.

2 Benchmarking Methodology for Ontology Tools

In the last decades, the word benchmarking has become relevant within the business management community. One of the definitions widely known was given by Spendolini [11] who defines benchmarking as a continuous, systematic process for evaluating the products, services, and work processes of organisations that are recognised as representing best practices for the purpose of organisational improvement.

The Software Engineering community does not have a common benchmarking definition. Some authors, like Kitchenham [12], consider benchmarking as a software evaluation method. For her, benchmarking is the process of running a number of standard tests using a number of alternative tools/methods and assessing the relative performance of the tools in those tests. Other authors, like Wohlin et al. [13], adopt the business benchmarking definition, defining benchmarking as a continuous improvement process that strives to be the best of the best through the comparison of similar processes in different contexts.

This section summarizes the benchmarking methodology developed by the authors in the Knowledge Web Network of Excellence [14]. The benchmarking methodology provides a set of guidelines to follow in benchmarking activities over ontology tools. This methodology adopts and extends methodologies of different areas such as business community benchmarking, experimental software engineering and software measurement as described in [14].

At the time of writing this paper, this methodology is being used in Knowledge Web for benchmarking the interoperability of ontology development tools.

Fig. 1 shows the main phases of the benchmarking methodology for ontology tools, which is composed of a benchmarking iteration that is repeated forever.

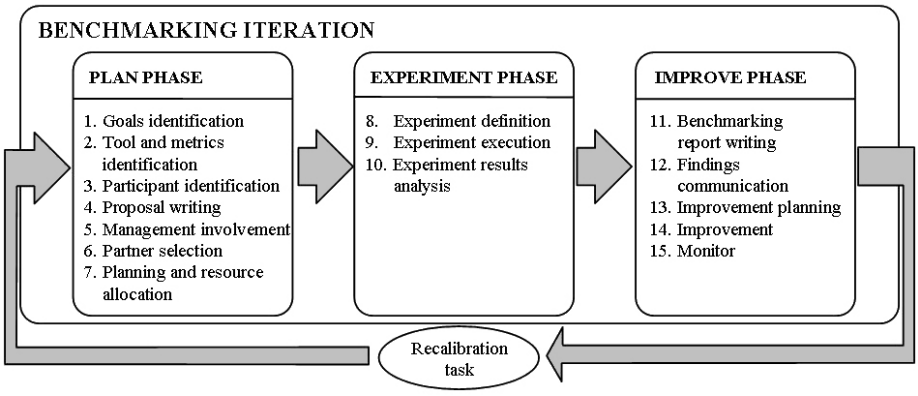


Fig. 1. Knowledge Web benchmarking methodology [14]

Each benchmarking iteration is composed of three phases (*Plan*, *Experiment* and *Improve*) and ends with a *Recalibration* task:

- **Plan phase.** Its main goal is to produce a document with a detailed proposal for benchmarking. It will be used as a reference document during the benchmarking, and should include all the relevant information about it: its goal, benefits and costs; the tool (and its functionalities) to be evaluated; the metrics to be used to evaluate these functionalities; and the people involved in the benchmarking. The last tasks of this phase are related to the search of other organisations that want to participate in the benchmarking with other tools, and to the agreement on the benchmarking proposal (both with the organisation management and with the other organisations) and on the benchmarking planning.
- **Experiment phase.** In this phase, the organisations must define and execute the evaluation experiments for each of the tools that participate on the benchmarking. The evaluation results must be compiled and analysed, determining the practices that lead to these results and identifying which of them can be considered as best practices.
- **Improve phase.** The first task of this phase comprises the writing of the benchmarking report, and this must include: a summary of the process followed, the results and the conclusions of the experimentation, recommendations for improving the tools, and the best practices found during the experimentation. The benchmarking results must be communicated to the participant organisations and finally, in several improvement cycles, the tool developers should perform the necessary changes to improve their tools and monitor this improvement.

While the three phases mentioned before are devoted to the improvement of the tools, the goal of the *Recalibration* task is to improve the benchmarking process itself using the lessons learnt while performing it.

3 Plan Phase

In this section we present the most relevant tasks from the *Plan* phase of the methodology. We will focus on those related to the identification of the benchmarking goals, the tool functionalities and the metrics; as these are the ones that influence the experimentation.

In order to evaluate the performance of ontology development tools, we make the assumption that these tools provide an ontology management API with methods to insert, update, remove, and query ontology components.

Therefore, our goal in the benchmarking is to **improve the performance of the methods provided by the ontology management APIs of the ontology development tools**.

For identifying the tool functionalities and metrics to be considered in the benchmarking, we have followed the Goal/Question/Metric (GQM) paradigm [15]. The idea beyond this is that any software measurement activity should be preceded by the identification of a software engineering goal, which leads to questions and that in turn lead to actual metrics. The questions and metrics derived from our goal are presented in Fig. 2. These questions and metrics show that the tool functionalities that are relevant in the benchmarking are the **methods** of the ontology management APIs, and that the metric to use is the **execution time** of the methods over incremental load states. After performing the experiments, the analysis of their results will provide answers to these questions.

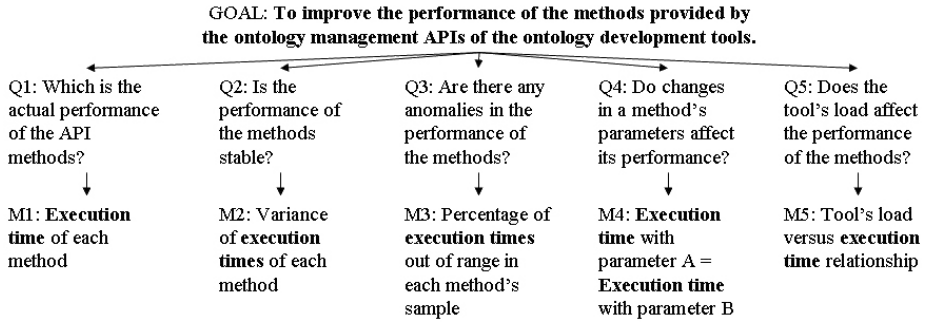


Fig. 2. Questions and metrics obtained through the GQM approach

4 Experiment Phase

This section presents the infrastructure needed when defining and executing experiments to evaluate the performance of the ontology management APIs of ontology development tools. We also identify the variables that influence the execution time of the methods and, in consequence, the evaluation results.

The **evaluation infrastructure** contains the different modules needed to achieve the benchmarking goal. Fig. 3 presents the main modules and the arrows represent the information flow between them.

These modules are described in the next sections, showing the main decisions taken regarding their design and implementation and giving examples according to the instantiation of the infrastructure for the WebODE ontology engineering workbench. In order to have a portable infrastructure, we have implemented it in Java, using only standard libraries and with no graphical components.

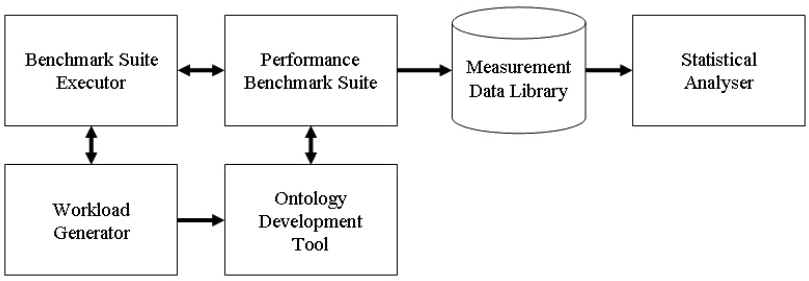


Fig. 3. Evaluation infrastructure

4.1 Performance Benchmark Suite

The Performance Benchmark Suite is a Java library that provides methods for executing each of the benchmarks that compose the benchmark suite. This benchmark suite should be developed taking into account the desirable properties of a benchmark suite [16, 17, 18, 19], that is, accessibility, affordability, simplicity, representativity, portability, scalability, robustness, and consensus.

In order to perform an evaluation of the complete system, every method in the ontology management API is present in the benchmark suite. For each of these methods, different benchmarks have been defined according to the changes in the methods' parameters that affect the performance.

The execution of the benchmarks is parameterised accordant with the parameter **number of executions** (N), which defines the number of consecutive executions of a method in a single benchmark whose execution times are measured. Moreover, the method is executed a certain number of times before starting the measurement so as to stabilise the ontology development tool.

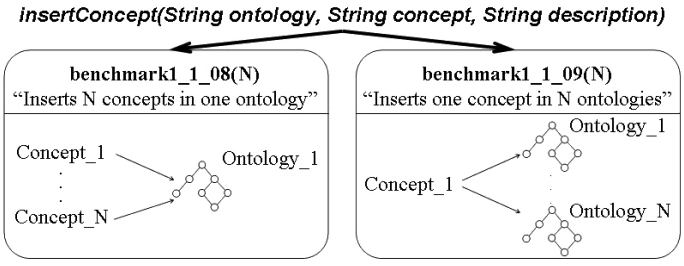


Fig. 4. Benchmarks defined for the method *insertConcept*

A benchmark executes just one method N times consecutively and stores in a text file the wall clock times elapsed in the method executions. The other operation performed by a benchmark is to restore the load state of the tool in case it changes during the benchmark execution.

In the case of WebODE, its ontology management API is composed of **72 methods**. From these methods, according to the different variations in their input parameters, we defined **128 benchmarks**[1].

For example, Fig. 4 shows the two benchmarks defined for the method *insertConcept* parameterized following the number of executions (N).

4.2 Workload Generator

The Workload Generator is a Java library that generates synthetic ontologies accordant with a predefined structure and to a load factor to insert them into the ontology development tool. The workload present in the ontology development tool must allow running the benchmarks with no errors and with different load factors.

The structure of the workload has been defined according to the execution needs of the benchmarks in order to run their methods a certain number of times (N) with no errors. For example, if a benchmark inserts one concept in N ontologies, these N ontologies must be present in the tool for a correct execution of the benchmark. Therefore, the execution needs of all the benchmarks in the benchmark suite define all the ontology components that must exist in the ontology development tool in order to execute every benchmark with no errors.

To define the workload independently of the number of executions of a method in a benchmark (N), we use a new parameter that defines the size of the ontology data. This is named the **load factor** (X) of the ontology development tool. With this load factor, we can define workloads of arbitrary size, but it must be taken into account that to execute the benchmark with no errors the load factor must be greater or equal to the number of executions of a method in a benchmark.

Hence, the workload used when executing all the benchmarks has the same structure as the execution needs of all the benchmarks but is parameterised to a load factor instead of to the number of executions of a method in a benchmark.

Table 1. Execution needs of the benchmarks whose methods insert and remove concepts

Benchmark	Operation	Execution needs
benchmark1_1_08	Inserts N concepts in an ontology	1 ontology
benchmark1_1_09	Inserts a concept in N ontologies	N ontologies
benchmark1_3_20	Removes N concepts from an ontology	1 ontology with N concepts
benchmark1_3_21	Removes a concept from N ontologies	N ontologies with one concept

Table 2. Execution needs of the benchmarks shown in Table 1

Benchmarks	Execution needs
benchmark1_1_08, benchmark1_1_09, benchmark1_3_20, and benchmark1_1_21	1 ontology with N concepts and N ontologies with1 concept

[1] http://kw.dia.fi.upm.es/wpbs/WPBS_benchmark_list.html

In the case of WebODE, Table 1 shows the execution needs of each of the four benchmarks whose methods insert and remove concepts in an ontology, being N the number of times that the method is executed. Table 2 shows the execution needs for executing the four benchmarks abovementioned with no errors.

4.3 Benchmark Suite Executor

The Benchmark Suite Executor is a Java application that controls the automatic execution of both the Workload Generator and the Performance Benchmark Suite.

This module defines the values of the variables that influence the evaluation: the one related to the infrastructure, that is, the ontology development tool's load factor (X); and the execution parameter of the benchmarks, that is, the number of executions (N).

The Benchmark Suite Executor guarantees that the load present in the ontology development tool allows executing the benchmarks with no errors (e.g. if a benchmark deletes concepts, these concepts must exist in the tool).

During the evaluation, the Benchmark Suite Executor performs two steps:

1. **To prepare the system for the evaluation.** It uses the Workload Generator for generating ontologies according to the load factor, and inserts them into the tool.
2. **To execute the benchmark suite.** It executes all the benchmarks that compose the benchmark suite. Each benchmark first stabilises the system by executing its corresponding method an arbitrary number of times, and then executes the method N more times, measuring the execution time. These N measurements of the execution time of the method are stored in a text file in the Measurement Data Library.

4.4 Measurement Data Library

The Measurement Data Library stores the results of the different benchmark executions. As the benchmarks provide their results in a text file, we do not propose a specific implementation for the Measurement Data Library.

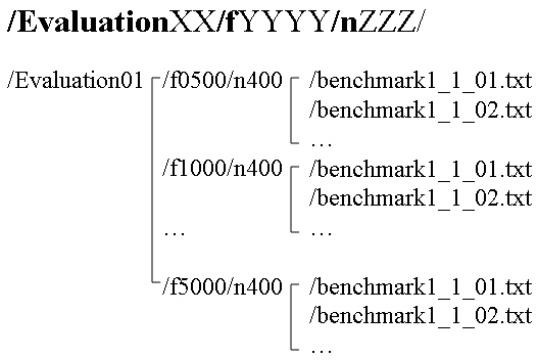


Fig. 5. Structure of the Measurement Data Library

The files with the results are stored in a hierarchical directory tree to be accessed easily. The structure of the tree, shown in Fig. 5, is the following:

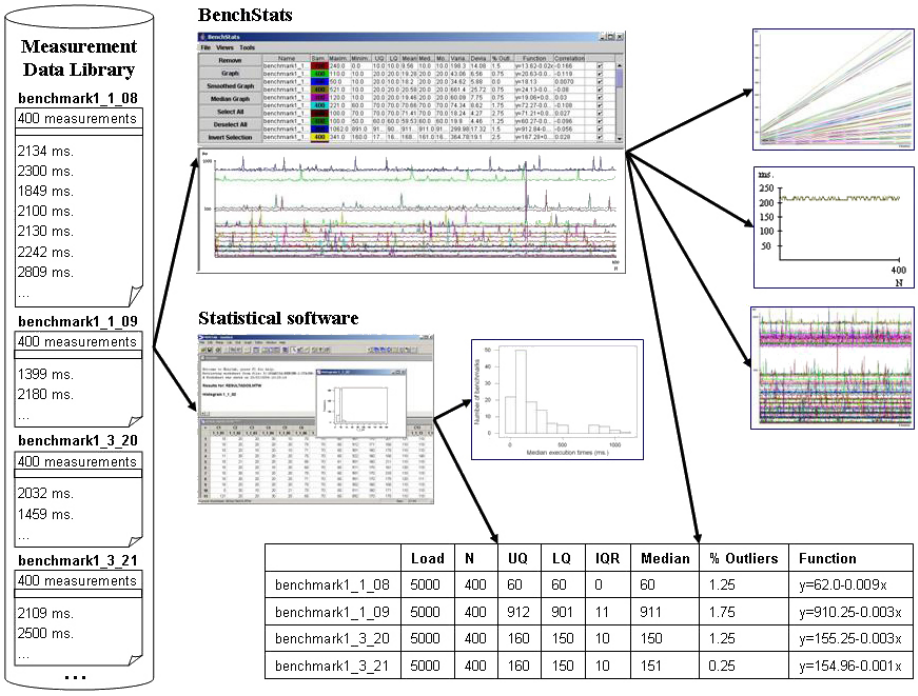
- A first level with the number of the evaluation (XX).
- A second level with the ontology development tool's load factor (YYYY).
- A third level with the number of executions of the benchmark (ZZZ).

4.5 Statistical Analyser

Any statistical tool can be used for analysing the results of the benchmarking. Nevertheless, a tool capable of automating parts of the analysis process, like report and graph generation, would facilitate the analysis of the results to a large extent.

As can be seen in Fig. 6, from the results of a benchmark stored in the Measurement Data Library, we can obtain different information that can be used to evaluate the ontology development tools:

- **Graphs** that show the behaviour of the execution times.
- **Statistical values** worked out from the measurements.



	Load	N	UQ	LQ	IQR	Median	% Outliers	Function
benchmark1_1_08	5000	400	60	60	0	60	1.25	y=62.0-0.009x
benchmark1_1_09	5000	400	912	901	11	911	1.75	y=910.25-0.003x
benchmark1_3_20	5000	400	160	150	10	150	1.25	y=155.25-0.003x
benchmark1_3_21	5000	400	160	150	10	151	0.25	y=154.96-0.001x

Fig. 6. Different information that can be extracted from the results

4.6 Variables that Influence the Execution Time

According to this evaluation infrastructure, there will be different variables that influence the execution time of a method. Some of them will be related to the features of the computer where the evaluation is performed (hardware configuration, software configuration and computer load) and one will be related to the infrastructure

proposed (the load of the ontology development tool). To compare the results of two benchmarks, they must be executed under the same conditions. The definitions of these variables are the following:

- **Hardware configuration.** It is the configuration of the hardware of the computer where the ontology development tool is running.
- **Software configuration.** It is the configuration of the operating system and of the software needed to execute the ontology development tool.
- **Computer load.** It is the load that affects the computer where the ontology development tool is running.
- **Ontology development tool load.** It is the amount of ontology data that the ontology development tool stores.

5 Evaluating WebODE's Ontology Management API

The *Experiment* phase of the benchmarking methodology comprises the evaluation of the tool once the evaluation infrastructure has been defined and implemented. According to the infrastructure presented in section 4, we defined the benchmark suite and implemented the necessary modules regarding WebODE and its ontology management API, and we performed the evaluation on WebODE.

From the different variables that affect the evaluation, we only considered changes in the tool's load variable, to know its effect in WebODE's performance. The other three variables took fixed values during the evaluation so they did not affect the execution times. Furthermore, to avoid other non-controlled variables that may affect the results, the computer used for the evaluation was isolated: it had neither network connection nor user interaction. Then, we defined the values that these variables took during the evaluation:

- **Hardware configuration.** The computer was a Pentium 4 2.4 Ghz monoprocessor with 256 Mb. of memory.
- **Software configuration.** Each software's default configuration was used: Windows 2000 Professional Service Pack 4; SUN Java 1.4.2_03; Oracle version 8.1.7.0.0 (the Oracle instance's memory configuration was changed to: Shared pool 30 Mb., Buffer cache 80 Mb., Large pool 600 Kb., and Java pool 32 Kb.); Minerva version 1 build 4; and WebODE version 2 build 8.
- **Computer load.** This load was the corresponding to the computer just powered on, with only the programs and services needed to run the benchmarks.
- **Ontology development tool load.** The benchmark suite was executed ten times with the following load factors: (X=500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, and 5000). As with a load factor of 5000 we obtained enough data to determine the methods' performance, the benchmarks have not been executed with higher load factors.

When running all the benchmarks in the benchmark suite:

- The method was first executed 100 times to stabilise the system before taking measures and to avoid unexpected behaviours in WebODE's initialisation.

- The **number of executions** (N) of a method in a benchmark was 400. With the aim of checking that 400 executions is a valid sample size, we have run several benchmarks with higher and lower number of executions and we have confirmed that the results obtained are virtually equivalent. We have not used a higher sample because the slightest precision improvement would mean a much higher duration of the benchmark suite execution.

After executing the 128 benchmarks of the benchmark suite with the 10 different load factors, we obtained 1280 text files, each with 400 measurements.

The source code of the infrastructure implemented for WebODE is published in a public web page[2], so anyone should be able to replicate the experiments and to achieve the same conclusions. The web page also contains the results obtained in this evaluation and all the statistical values and graphs worked out from them.

6 Analysis of the Evaluation Results

We have regarded the results of executing the benchmark suite with the maximum load factor used (X=5000) to be able to clearly differentiate the execution times. When analysing the effect of WebODE's load in the execution times of the methods, we have considered the results of executing the benchmark suite from a minimum load state (X=500) to a maximum load state (X=5000). In every case, we have considered a number of executions (N) of 400.

A first rough analysis of the results of the benchmark suite execution showed two main characteristics:

- Observing the graphs of the execution times measured in a benchmark, we saw that execution times are mainly **constant**. This can be seen in Fig. 7 that shows the execution times of running the method *removeConcept* 400 times with a load factor of 5000 in benchmark1_3_20.
- After running normality tests over the measurements, we confirmed that the distributions of the measurements were non-normal. Therefore, we could not rely on usual values such as mean and standard deviation for describing them and thus we used **robust statistical values** like the median, the upper and lower quartiles, and the interquartile range (upper minus lower quartile).

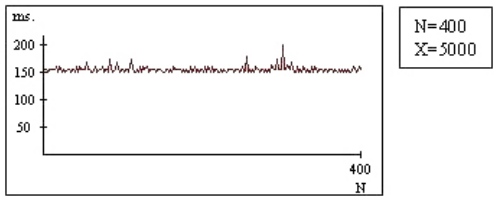


Fig. 7. Execution times of *removeConcept* in benchmark1_3_20

[2] http://kw.dia.fi.upm.es/wpbs/

The next sections show the specific metrics used for analysing the performance of the methods and the conclusions obtained from the execution results, that answer the questions previously stated in Fig. 2.

6.1 Metric for the Execution Time

The metric used for describing the execution time of a method in a benchmark has been the **median** of the execution times of the method in a benchmark execution.

Fig. 8 shows the histogram of the medians of the execution times of all the API methods. These medians range from 0 to 1051 milliseconds, with a group of values higher than the rest. The medians in this group belong to 12 benchmarks that execute 8 methods (as different benchmarks have been defined for each method). These 8 methods, with a median execution time higher than 800 ms, have been selected for the improvement recommendations. The rest of the median execution times of the methods are lower than 511 ms, being most of them around 100 ms.

Bearing in mind the kind of operation that the methods carry out (inserting, updating, removing, or selecting an ontology component), we did not find significant differences between the performances of each kind of method.

Taking into account what kind of element of the knowledge model (concepts, instances, class attributes, instance attributes, etc.) a method manages, in the slowest group almost every method that manages relations between concepts are present. Methods that manage instance attributes also have high execution times, and the rest of the methods behave similarly; the methods that stand out are those that manage imported terms and references since they are the ones with lower execution times.

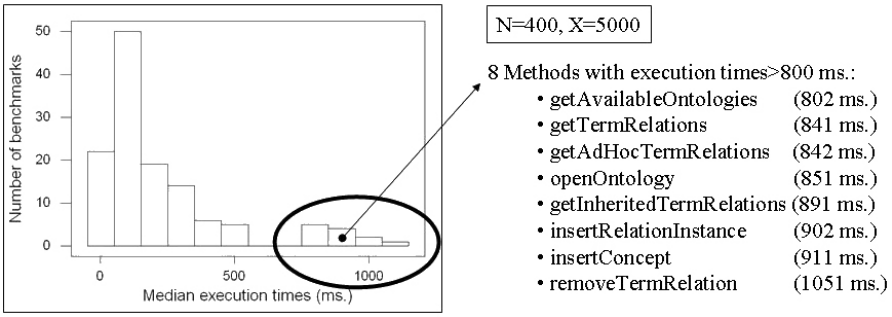


Fig. 8. Histogram of the medians of the execution times

6.2 Metric for the Variability of the Execution Time

The metric used for describing the variability of the execution time of a method in a benchmark has been the **interquartile range** (IQR) of the execution times of the method in a benchmark execution.

Fig. 9 shows the histogram of the IQRs of the execution times. Almost every method has an IQR from 0 to 11 ms, which is a low spread considering that the granularity of the measurements is 10 milliseconds. The only exceptions are the three methods shown in the figure. The method *getAvailableOntologies* has been selected for the improvement recommendations because of its atypical IQR value.

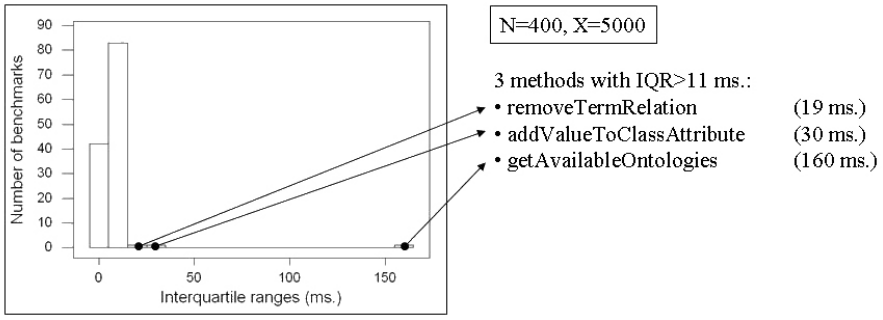


Fig. 9. Histogram of the interquartile ranges of the execution times

6.3 Metric for the Anomalies in the Execution Time

The metric used for describing the anomalies in the execution time of a method in a benchmark has been the **percentage of outliers** in the execution times of the method in a benchmark execution.

The traditional method for calculating the outliers is to consider as potential outlier values the measurements beyond the upper and lower quartiles and to add and subtract respectively 1.5 times the interquartile range [20]. As the Java method used for measuring time (*java.lang.System.currentTimeMillis()*) in the Windows platform has a precision of tens of milliseconds, in the results we frequently encountered interquartile ranges of zero milliseconds. This made us consider as outliers every determination that differed from the median. With the objective of fixing this precision fault, we have augmented the interquartile range when calculating the outliers to include half the minimal granularity (5 milliseconds) in both boundaries.

Fig. 10 shows the histogram of the percentage of outliers in the execution times of the methods. Most of the benchmarks range from 0 to 3.75% of outliers. These values confirm the lack of anomalies except the peaks in the execution times that can be seen in the graphs. The only two methods to emphasize, shown in the figure, have been selected for the improvement recommendations.

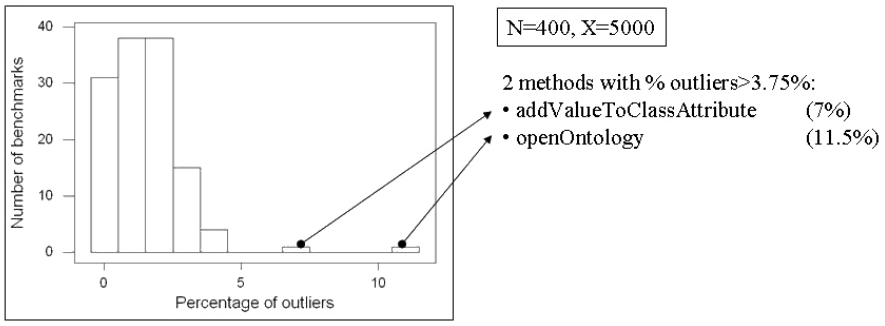


Fig. 10. Histogram of the percentage of outliers of the execution times

6.4 Effect of Changes in the Parameters of a Method

To analyse if changes in the parameters of a method affect the method performance, we compared the medians of the execution times of the benchmarks that use the same method.

The performance of 21 methods varies when its input parameters are changed, but this variation is lower than 60 milliseconds except in the five methods shown in Fig. 11, that have been selected for the improvement recommendations. Fig. 11 also shows the comparison of the execution times of the method *insertConcept* in benchmark1_1_08 and in benchmark1_1_09.

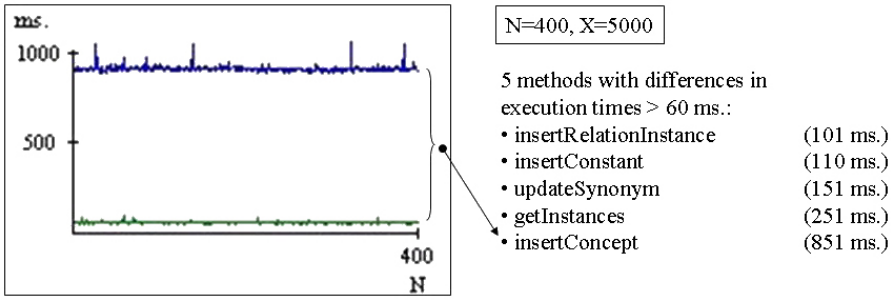


Fig. 11. Execution times of *insertConcept* in benchmark1_1_08 and benchmark1_1_09

6.5 Effect of Changes in WebODE's Load

To analyse the effect of WebODE's load in the execution times of the methods, we studied the medians of the execution times of the methods from a minimum load state (X=500) to a maximum load state (X=5000). We estimated the function that these medians define by **simple linear regression** and considered its **slope** in order to examine the relationship between the load and the execution time of the methods.

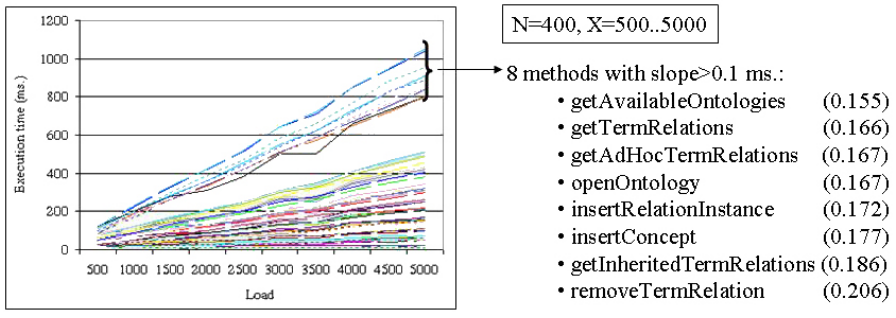


Fig. 12. Evolution of the execution times when increasing WebODE's load

Fig. 12 shows, for every benchmark, the functions defined by the median execution times with the different load factors. The slopes of the functions range from 0 to 0.1

except in 8 methods. The 8 methods whose execution times are higher than the rest are also the methods whose performance is more influenced by the load, and have been selected for the improvement recommendations.

6.6 Improvement Recommendations

From the analysis of the results, we produced a report stating the recommendations to improve WebODE's performance. These recommendations include the methods of the WebODE ontology management API identified in the previous sections.

Table 3 shows a summary of the improvement recommendations with 12 of the 72 WebODE's API methods included in them, and the reasons for their inclusion.

Table 3. Methods in the improvement recommendations

	Execution time > 800 ms	Interquartile range > 150 ms	Outlier values > 3.75%	Execution time variation > 60 ms	Slope when increasing load > 0.1
removeTermRelation	X				X
getInheritedTermRelations	X				X
insertConcept	X			X	X
insertRelationInstance	X			X	X
openOntology	X		X		X
getAdHocTermRelations	X				X
getTermRelations	X				X
getAvailableOntologies	X	X			X
addValueToClassAttribute			X		
insertConstant				X	
updateSynonym				X	
getInstances				X	

7 Conclusions and Future Work

In this paper we provide an overview of the benchmarking methodology for ontology tools developed by the authors in Knowledge Web. We define some guidelines when using this methodology to improve the performance and the scalability of ontology development tools by evaluating the performance of their ontology management APIs' methods.

To support the experimentation tasks of the methodology, we provide a detailed definition of an infrastructure for evaluating the performance and the scalability of ontology development tools' ontology management APIs. We have instantiated this infrastructure for evaluating the ontology management API of the WebODE ontology engineering workbench and the results obtained after the evaluation provide us with precise information on WebODE's performance.

The evaluation infrastructure can be instantiated for evaluating other ontology development tools that provide ontology management APIs. Taking as a starting point the methods of the ontology management API of a certain tool, the following tasks should be performed:

- Benchmarks that evaluate these methods should be defined, and the Performance Benchmark Suite module should be implemented for executing them.
- The Workload Generator should also be implemented to generate workload according to these methods' needs.
- The rest of the modules (Benchmark Suite Executor, Measurement Data Library and Statistical Analyser) already instantiated for WebODE could be used for another tool with minimal or no changes.

To obtain all the benefits of the benchmarking, like the extraction of best practices, other ontology development tools should participate in it. In this case, there are other tasks of the methodology that should be considered and that are not covered by this paper such as the search of other organisations and tools for participating in the benchmarking, the planning of the benchmarking, and the improvement on the tools. To perform a benchmarking like this, the evaluation infrastructure must be the same for every tool. Therefore:

- The Workload Generator should be modified in order to generate workloads independent of the tool, and thus the same workload can be used for every tool.
- The Performance Benchmark Suite should be modified to include only the methods common to all the tools or to use a common ontology management API such as OKBC [21].

Although the benchmark suite execution is automatic, the evaluation infrastructure would benefit significantly if some automatic analysis and summary of the results could be carried out, as there are plenty of them.

The WebODE Workload Generator could be improved and could generate ontologies with other structure or characteristics. In consequence, this module could be employed in other kind of evaluations and, thanks to the WebODE export services to different formats and languages (like RDF(S) or OWL); these ontologies could be used in evaluations performed over other tools, not just over WebODE.

Acknowledgments

This work is partially supported by a FPI grant from the Spanish Ministry of Education (BES-2005-8024), by the IST project Knowledge Web (IST-2004-507482) and by the CICYT project Infraestructura tecnológica de servicios semánticos para la web semántica (TIN2004-02660). Thanks to Rosario Plaza for reviewing the grammar of this paper.

References

1. A.J. Duineveld, R. Stoter, M.R. Weiden, B. Kenepa, and V.R. Benjamins. Wondertools? a comparative study of ontological engineering tools. In Proceedings of the 12th International Workshop on Knowledge Acquisition, Modeling and Management (KAW'99), Banff, Canada, 1999. Kluwer Academic Publishers.
2. Ontoweb deliverable 1.3: A survey on ontology tools. Technical report, IST OntoWeb Thematic Network, May 2002.

3. P. Lambrix, M. Habbouche, and M. Pérez. Evaluation of ontology development tools for bioinformatics. Bioinformatics, 19(12):1564-1571, 2003.
4. J. Angele and Y. Sure (eds.). Proceedings of the 1st International Workshop on Evaluation of Ontology-based Tools (EON2002), Sigüenza, Spain, September 2002.
5. Y. Sure and O. Corcho (eds.). Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
6. Y. Sure, O. Corcho, J. Euzenat, T. Hughes (eds.). Proceedings of the 3rd International Workshop on Evaluation of Ontology-based Tools (EON2004), Hiroshima, Japan, November 2004.
7. A. Magkanaraki, S. Alexaki, V. Christophides, and D. Plexousakis. Benchmarking RDF schemas for the semantic web. In Proceedings of the First International Semantic Web Conference, pages 132–146. Springer-Verlag, 2002.
8. C. Tempich and R. Volz. Towards a benchmark for semantic web reasoners - an analysis of the DAML ontology library. In Proc. of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003), Florida, USA, October 2003.
9. Y. Guo, Z. Pan, and J. Heflin. An evaluation of knowledge base systems for large OWL datasets. In Proceedings of the 3rd International Semantic Web Conference (ISWC2004), pages 274.288, Hiroshima, Japan, November 2004.
10. J.C. Arpírez, O. Corcho, M. Fernández-López, A. Gómez-Pérez. WebODE in a nutshell. AI Magazine. 24(3), Fall 2003, pp. 37-47.
11. M. Spendolini. The Benchmarking Book. AMACOM, New York, NY, 1992.
12. B. Kitchenham. DESMET: A method for evaluating software engineering methods and tools. Technical Report TR96-09, Department of Computer Science, University of Keele, Stanfordshire, UK, 1996.
13. C. Wohlin, A. Aurum, H. Petersson, F. Shull, and M. Ciolkowski. Software inspection benchmarking - a qualitative and quantitative comparative opportunity. In Proceedings of 8th International Software Metrics Symposium. 118-130, 2002.
14. R. García-Castro, D. Maynard, H. Wache, D. Foxvog, and R. González-Cabero. D2.1.4 Specification of a methodology, general criteria and benchmark suites for benchmarking ontology tools. Technical report, Knowledge Web, December 2004.
15. V.R. Basili, G. Caldiera, D.H. Rombach. The Goal Question Metric Approach. Encyclopedia of Software Engineering, 2 Volume Set Willey, pp 528-532, 1994.
16. J.M. Bull, L.A. Smith, M.D. Westhead, D.S. Henty, R.A. Davey. A Methodology for Benchmarking Java Grande Applications. EPCC, June 1999.
17. B. Shirazi, L. Welch, B. Ravindran, C. Cavanaugh, B. Yanamula, R. Brucks, E. Huh. DynBench: A Dynamic Benchmark Suite for Distributed Real-Time Systems. IPDPS Workshop on Embedded HPC Systems and Applications. S. Juan, Puerto Rico, 1999.
18. S. Sim, S. Easterbrook, and R. Holt. Using benchmarking to advance research: A challenge to software engineering. In Proceedings of the 25th International Conference on Software Engineering (ICSE'03), pages 74-83, Portland, OR, 2003.
19. F. Stefani, D. Macii, A. Moschitta, and D. Petri. FFT benchmarking for digital signal processing technologies. In 17th IMEKOWorld Congress, Dubrovnik, June 2003.
20. W. Mendenhall and T. Sincich. Statistics for Engineering and the Sciences, 4th Edition. Englewood Cliffs, NJ. Prentice Hall, 1995.
21. V.K. Chaudhri, A. Farquhar, R. Fikes, P.D. Karp, J.P. Rice. The Generic Frame Protocol 2.0. Technical Report, Stanford University, 1997.