Towards a benchmark of the ODE API methods for accessing ontologies In the WebODE platform

Oscar Corcho, Raúl García-Castro, Asunción Gómez-Pérez
(Ontology Group, Departamento de Inteligencia Artificial, Facultad de Informática,
Universidad Politécnica de Madrid, Spain
{ocorcho@fi.upm.es, rgarcia@delicias.dia.fi.upm.es, asun@fi.upm.es})

Abstract: Ontology editors and ontology engineering platforms allow creating and maintaining ontologies and using them in a wide range of applications, but there are neither specific benchmark for evaluating ontology platforms nor for evaluating their ontology access services. In this paper we present how we have designed and structured a benchmark for the ontology access services of the WebODE platform. We also present some results and analysis of the benchmark suite execution.

1 Introduction

In recent years, the use of ontology editors and ontology engineering platforms is spreading and the size of the ontologies they store is growing. Benchmarking is a process of continuously measuring and comparing a system's components against other systems components to gain information which will help the organization take action to improve its performance [Alstete, 1992]. There are plenty of benchmarking studies in other fields like database or compilers. However, there aren't specific benchmarks studies or tools for evaluating ontology editors and platforms.

As presented in [Bull et al., 2000], a benchmark suite must be ***representative*** (reflecting all the types of functions used in the tool), ***simple*** (being clear what is being tested and with understandable results), ***robust*** (being non-sensitive to external factors), ***portable*** (running on a wide variety of environments), and ***standardised*** (having a public common structure).

In this paper we present how we have carried out the benchmark of the public ontology management methods used by the WebODE platform [Arpírez et al., 03] for accessing their ontologies. As these functions are present in most of the ontology editors and platforms, this benchmark suite can be considered as a first approach to the design of a benchmark suite for ontology tools. Besides, we show the results and the conclusions obtained from the analysis of these results.

The paper is structured as follows. Section 2 presents the main goals of the benchmark suite and explains the four parameters that will be analysed. Section 3 describes the tests designed: how the test groups are structured, and their initial states. In section 4 we present the main results of running all these tests. Section 5 concludes with the analysis of the previous results. Finally, section 6 presents future work.

2 Goals of the benchmark suite design and execution

WebODE is a scalable workbench for ontological engineering that eases the design, development and management of ontologies and includes *middleware services* to aid

in the integration of ontologies into real-world applications. WebODE presents an ideal framework to integrate new ontology-oriented tools and services, where developers only worry about the new logic they want to provide on top of the knowledge stored in their ontologies. In this platform the ODE API is a crucial piece of code that contains all the public methods of the WebODE ontology access service. The majority of the services provided by the platform use the ODE API methods for accessing the ontologies. So, it is obvious that a benchmark of the ODE API is a must before benchmarking more advanced and complex services. In order to achieve portability, we have developed the benchmark suite in Java, using standard methods and no graphic components.

The ODE API includes for the time being 72 public methods mainly related to the update, removal and insert of content into the ontologies. In this paper we present the results of benchmarking the performance of the public methods of the ODE API. Other public methods used for exporting, importing, merging, evaluating and reasoning with the ontologies will be benchmarked in further papers.

According to the different possible input parameter values that these methods can have, we have defined a number of tests. For instance, for the method that allows inserting concepts in an ontology: *void insertConcept (String ontology, String term, String description)* we have defined two tests named:

B1_1_08: Inserts concepts into the same ontology.

B1_1_09: Inserts concepts into different ontologies.

As a consequence, we have defined 128 different tests for the 72 public methods of the ODE API.

With regard to the performance metrics used, we have decided only to measure on the initial stages of the benchmark the ***wall clock time*** of each public method in the following situations:

- ***Running in a high load state***. These results will allow analysing the performance of the public methods in a high load condition.
- ***Running several times over the same load state***. This analysis will provide clues about the public method's performance over the time.
- ***Running on different increasing load states***. These results are useful to know the method's sensitiveness to the load states under which it is run.
- ***Running with different input parameters***. This analysis will show the sensitiveness of each method's performance with regards to its input parameters.

3 Description of the benchmark suite

Once we have analysed the requirements for our benchmark suite and the performance metrics that we will use for it, we present how we have grouped the existing methods on the ODE APIs and the initial load state for all the tests.

3.1 Structure of the tests

The tests included are classified into six groups with the aim of having a representative and interpretable set of tests in our benchmark suite [Williams et al,

2002]. The test groups were identified just looking similarities on the functionality of the methods provided by the ODE API, and they are:

- ***Inserts***. Includes tests whose methods insert new ontologies or new content in an ontology. This group contains 37 tests, related to 19 methods.
- ***Updates***. Includes tests whose methods update information of ontologies or content in an ontology. This group contains 18 tests, related to 9 methods.
- ***Removals***. Includes tests whose methods remove the whole ontologies or some content of the ontology. This group contains 21 tests, related to 11 methods.
- ***Selects***. Deals with tests whose methods select ontologies or ontology content. This group contains 47 tests, related to 30 methods.
- ***Non basics***. Groups tests whose methods use other methods from the ODE API. This group contains 5 tests, related to 3 methods.
- ***Complex***. Groups tests for benchmarking methods and services that are not included in the ODE API. Such tests are associated to methods that perform complex operations in the platform, and are mainly related to ontology import and export, inferences, ODEClean evaluation, merge, etc. This group contains 17 tests, related to 15 methods.

On a first step, we are only dealing with the first five groups of tests, and we have postponed the last group for further benchmarkings.

3.2 Initial states

As we mentioned before, robustness is an important feature identified in Bull and colleagues's work. To achieve a higher level of robustness, all the tests must be compared according to similar situations. Because of that, we have defined clearly the initial state of the computer where the tests are executed and the initial load state of the whole benchmark suite. Both of them are minimal and common for all the tests.

For each individual test we have defined its initial load state. For instance, when executing the test B1_2_11 ("update synonyms in a concept") with respect to a load factor X, the system must have one ontology with one concept with X synonyms.

The initial load state of the benchmark suite is the union of the initial load states of each individual test. So, the composed initial state of the benchmark suite is: one ontology with X references; X constants; X formulas; one concept with X class attributes and X instance attributes; X concepts with one class attribute and one instance attribute, etc.; and X ontologies with one reference, one constant, one formula, two concepts with one class attribute and one instance attribute, etc.

4 Execution of the benchmark suite

To execute the tests, we have taken into account the following two parameters:

- ***Load factor***. This parameter sets the load factor X for each test's initial state, as described in the previous section. We have used factors from 10 to 5000.
- ***Number of iterations***. This parameter sets the number of consecutive executions of the test. We have used iterations from 10 to 5000 to detect different possible behaviours in the execution.

In order to retrieve and store the data to be used for the subsequent analysis, we have executed each test with all the different load factors and number of iterations.

5 Analysis of results of the execution

After executing the tests, the results were analysed using common statistical analysis parameters such as medians, variances, etc., and common hypothesis tests.

We have to bear in mind that the results of executing a benchmark suite are usually temporary limited [Gray, 1993]. As the methods in the ODE API will undergo changes, the results of the current benchmark just inform us about WebODE current performance. So, if changes happen on the ODE API, the current results are not valid any more and the re-execution of this benchmark suite will be necessary again.

The following figures are samples of the different types of results obtained through the execution of the tests. Figure 1 presents the execution time evolution for all the insert tests with the highest load, and Figure 2 shows the execution times evolution for the test B1_1_18 ("add value to a instance attribute") with different loads. These figures show the results in milliseconds.

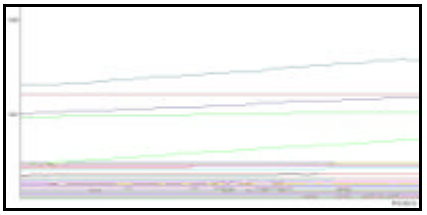

Figure 1. Execution times for the insert tests

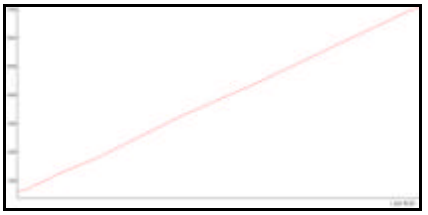

Figure 2. Execution times evolution for the test B1_1_18

The goal of analysing the results of **running tests with a high load state** is to detect the slowest and fastest methods of the ODE API, so that we can decide which ones to optimise in order to improve the WebODE performance. Here, we have considered the median of the test execution time with the highest load state (X=5000). In general, regarding the ODE API methods, the slower methods are those which manage attribute values and instances.

To analyse the results of **running tests several times over the same load state**, we have used the highest load (X=5000), and the highest number of iterations (N=5000) in order to have more complete results. For each function, we have analysed how its execution times evolve over time when it is called several times consecutively. A general comment for these tests is that the execution time is usually constant, no matter how many times the test is performed. There are some situations where the previous statement is not true, as in the case of some insertions: when the number of elements in the system increases the execution time also increases. In the case of some removals, the opposite effect can be observed. Figure 1 shows the results of all the tests of the insert group described above.

In order to analyse what happens when **running tests on different load states**, we also use the medians of the execution times of each function according to each load state with which we have performed the tests. Then we try to approximate them to linear or quadratic functions. The overall results of these tests are that all of them can be linearly estimated. The slope of the functions varies according to the execution time of each test: the slower a method, the greater the slope of its function. In figure 2

we show the function obtained for the test B1_1_18 ("insert values in an instance attribute") which can be easily approximated to a linear function: $Y = 30 + 0.136X$.

The aim of analysing the results of **running tests with different parameters** is to show the sensitiveness of each function to the modification of its input parameters. In general, changing a method's parameters doesn't affect significantly to the performance in the ODE API except in some cases where different behaviour can be detected when a function's input parameters are different.

6 Future Work

The analysis performed in this work is just based on a single performance metric: the execution time of each public method of the ODE API. As part of our future work we will aim at the creation of better performance metrics that allow assessing more easily the overall performance of the ODE API and more complex services built on top of it.

Although this analysis has proven useful to detect bottlenecks in the system another extension to this benchmark consists of synthetically studying the performance of the applications using these ontology services.

Running the benchmark suite and analysing its results is highly time consuming. Because of this, another future step consists on modifying the benchmark suite so that it just focuses on certain relevant issues considered critical.

Finally, this benchmark suite has been designed specifically for the WebODE ontology engineering workbench. We plan to extend our benchmark suite to other platforms, either by finding commonalities between the ontology access APIs of the different platforms or by forcing all these platforms to share a common API such as OKBC [Chaudhri et al., 1997].

Acknowledgements

This work is partially supported by a FPU grant from the Spanish Ministry of Education (AP2002-3828), and by the IST project Esperonto (IST-2001-34373).

References

[Alstete, 1992] Alstete, J. W. *Competitive Benchmarking Course.* Technical Report, 1992 http://www.iona.edu/faculty/jalstete/MNG992/documents.htm

[Arpírez et al., 2003] Arpírez JC, Corcho O, Fernández-López M, Gómez-Pérez A. *WebODE in a nutshell.* AI Magazine. To appear in Fall 2003.

[Bull et al., 2000] Bull JM, Smith LA, Westhead MD, Henty DS, Davey RA. *A Benchmark Suite for High Performance Java.* Concurrency: Practice and Experience (12):375-388, 2000.

[Chaudhri et al., 1997] Chaudhri V. K.; Farquhar A.; Fikes R.; Karp P. D.; Rice J. P. *The Generic Frame Protocol 2.0.* Technical Report, Stanford University.1997.

[Gray, 1993] Gray, J. The Benchmark Handbook for Database and Transaction Systems (2nd Edition). Morgan Kaufmann 1993

[Williams et al, 2002] Williams, LG., Smith, CU. Five Steps to Solving Software Performance Problems. http://www.perfeng.com 2002