Building an Efficient Component for OCL Evaluation ⋆

Manuel Clavel[1,2], Marina Egea[1,2], and Miguel A. García de Dios[2]

[1] IMDEA Software Institute, Madrid, Spain
[2] Universidad Complutense de Madrid, Madrid, Spain.
[manuel.clavel,marina.egea]@imdea.org,miguelgd@fdi.ucm.es

Abstract. In this paper we report on our experience developing the Eye OCL Software (EOS) evaluator, a Java component for efficient OCL evaluation. We first motivate the need for an efficient implementation of OCL in order to cope with novel usages of the language. We then discuss some aspects that, based on our experience, should be taken into account when building an OCL evaluator for medium-large scenarios. Finally, we explore various approaches for evaluating OCL expressions on really large scenarios.

1 Motivation

In the recent past we have worked on the definition of a formal and executable semantics for the Object Constraint Language (OCL) [20]. The results of this research will soon appear in a doctoral dissertation [11]. They also provide the foundations of the ITP/OCL tool [6], a rewriting-based evaluator for OCL expressions on instances of user-defined models. As part of our research, we have looked at different usages of OCL beyond its "initial requirements as a precise modeling language complementing UML specifications." Two related applications have drawn our interest [10, 2, 3, 7, 4], both having to do with using OCL to analyze user-defined models by evaluating queries on the corresponding instances of their metamodels. Since these instances typically contain a large number of elements, evaluating expressions on them comes at a high computational cost.

Consider, for example, the use of OCL to express metrics for Java programs (this application was suggested to us by members of the Triskell group at IRISA, France). The scenarios on which the program metrics will be evaluated are the instances of the Java metamodel corresponding to the programs: thus, the larger the programs the larger the scenarios[3] and, consequently, the higher the computational cost of evaluating the program metrics.

⋆ Research supported by Spanish MEC Project TIN2006-15660-C02-01 and by Comunidad de Madrid Program S-0505/TIC/0407.

[3] An an example, the SpoonEMF application, developed by the Triskell group, generates, for a standard Java program with 10 lines, a scenario with 113 objects; for a program with 100 lines, one with 1180 objects; and for a program with 500 lines, one with 3470 objects.

We report here on our experience developing the Eye OCL Software [8] (EOS) component, an OCL evaluator designed with the goal of performing efficient evaluation of OCL expressions on medium-large size scenarios. In particular, we discuss i) the need for an efficient implementation on OCL in order to cope with the novel usages of the language; ii) the aspects that we have taken into consideration to improve the efficiency of the EOS evaluator on medium-large scenarios; iii) the limits of the current OCL implementations for dealing with really large scenarios. Although we include the results of applying a benchmark to several OCL evaluators, this paper is not a comparative study (see [12] for a recent study of this kind). In fact, the results are included here only to show the current performance of some OCL tools on medium-large scenarios and to illustrate the aspects that we consider that should be taken into account when implementing an efficient OCL evaluator for medium-large scenarios. Interestingly, this quality —OCL engine efficiency on medium-large scenarios— is not covered by the benchmark proposed in [12]: in fact, the largest scenarios scenario proposed for testing OCL engine efficiency in [13] contains 42 objects and 42 links. Furthermore, despite the results of our benchmark, this paper is not a promotional brochure for our EOS component: as a "product", our OCL evaluator is still in its infancy; the fact is that we have only worked a few months in its implementation, which is rather straightforward except for those aspects that are explicitely discussed in this paper.

To motivate the need for OCL engines that can efficiently evaluate expressions on medium-large size scenarios, we show in Table 1 the time that currently takes to evaluate two given OCL expressions on three different, small-medium size scenarios for a number of OCL evaluators: namely, USE 2.4.0 [15], RoclET [1], OCLE [5], MDT OCL [19], and EOS [8].[4] The tests were run on a laptop computer, with Windows XP Professional installed, a processor Intel Pentium M 2.00GHz 600MHz, and 1GB of RAM. Also, in the case of the EOS and USE evaluators, we run the JVM with its parameters `-Xms` and `-Xmx` set to 1024m.

The scenarios considered in these tests are instances of the model Library shown in Figure 1: each scenario is referenced by a number n, which also indicates its "size"; more precisely, for each n, the scenario $\#n$ is a library that contains exactly 10^n books, each book with a unique title different from "Hobbit". The OCL expressions used in these tests are

$$\mathsf{Book.AllInstances()}{-}{>}\mathsf{forAll(b|b.title} <> \mathsf{'Hobbit')} \tag{1}$$

$$\mathsf{Book.AllInstances()}{-}{>}\mathsf{forAll(b1,b2|b1} <> \mathsf{b2\ implies\ b1.title} <> \mathsf{b2.title).} \tag{2}$$

The first expression says that the library does not contain any book titled "Hobbit", while the second one says that the library does not contain two different

[4] In the case of USE, we have run our experiments using its version 2.4.0. For MDT OCL, we have run the experiments using the plug-in "OCL Interpreter", version 1.2.0v200805130238. Finally, for our experiments in OCLE, we have used its version 2.0.

books with the same title. Obviously, the cost of evaluating these expressions depends on the number of books in the library and the cost of accessing, and storing for later use, information about these books. For example, in order to evaluate the expressions (1) and (2) on scenario #3 we have to perform, respectively, 10^3 and $2 \times 10^3 \times 10^3$ times the operations of accessing and storing a book's title. But this is precisely one of the challenges for OCL engines when evaluating expressions on medium-large size scenarios: namely, to efficiently access the information contained in, possibly, all the objects that populate the scenarios.

Notice that in Table 1 we use $> t$ to indicate that we stopped an experiment after having passed time t without obtaining the result. Also, we use a dash to indicate that we did not run the experiment, because we already stopped the execution of a "simpler" one. Finally, since RoclET, OCLE and MDT OCL do not report their execution time in milliseconds, we use $<$ 1s to indicate that the result of an evaluation was output in less than 1 second, and we use $\approx t$ to indicate that the result of an evaluation was output approximately in time t.

Scenario	*Expression*	RoclET	OCLE	MDT OCL	EOS	USE
#2	(1)	$<$ 1s	$<$ 1s	$<$ 1s	0ms	230ms
	(2)	$>$ 10m	$\approx$ 4s	$<$ 1s	50ms	240ms
#3	(1)	–	$\approx$ 3s	$<$ 1s	10ms	240ms
	(2)	–	$>$ 10m	$\approx$ 4s	841ms	1s342ms
#4	(1)	–	–	$<$ 1s	20ms	261ms
	(2)	–	–	$\approx$ 5m12s	1m18s	2m12s

Table 1. Evaluating performance on medium-large scenarios.

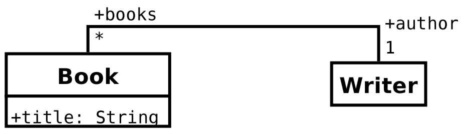

Fig. 1. The model Library.

2 Measuring the cost of evaluating expressions

Although the computational cost of evaluating a particular OCL expression on a given scenario obviously depends on the algorithms and data structures used to implement each tool, based on our experience, there are two measurements worthwhile considering before launching the evaluation process: first, the maximum number of times that objects' properties will be accessed and, second, the maximum size of the collections that will be built. In the case of medium-large size scenarios, the challenge for OCL engines is that these measurements typically return large numbers.

To illustrate this challenge, we show in Table 2 the performance of MDT OCL, EOS, and USE when evaluating different iterator-expressions whose evaluation require accessing many times objects' properties and/or building large collections. All the expressions in Table 2 were evaluated on the same scenario, namely, an instance MyLibrary of the model Library shown in Figure 1 which contains 10^3 authors, each author with 10 different books, and each book with a title different from "Hobbit".[5] For the sake of the experiment, we artificially increased the size of the collections to be iterated upon: in particular, in Table 2, the terms p_1, p_2, and p_3 refer, respectively, to the expressions "Book.allInstances().author.books", "p_1.author.books", and "p_2.author.books", which on the scenario MyLibrary evaluate to collections with 10^5, 10^6, and 10^7 books, respectively.[6] The iterator-expressions in Table 2 were evaluated on a laptop computer, with Windows XP Professional installed, a processor Intel Pentium M 2.00GHz 600MHz, and 1GB of RAM. Also, in the case of the EOS and USE evaluators, we run the JVM with its parameters `-Xms` and `-Xmx` set to 1024m. For the reason explained above, in the case of the MDT OCL evaluator all times shown in Table 2 are approximated ($\approx$).[7] The *Error* evaluation time indicates that we could not evaluate the expression due to an OutOfMemoryError reported by Java.

Let $exp(p_i)$, with $i \in \{1, 2, 3\}$, be the time that takes to evaluate an iterator-expression exp on the collection generated by evaluating the term p_i. With respect to the performance of MDT OCL, EOS, and USE, Table 2 shows that, for the same exp, the time $exp(p_{i+1})$ is approximately $10 \times exp(p_i)$ in these tools. Table 2 also shows that, for the same p_i, the time $exp(p_i)$ depends for these tools on the number of accesses to objects' properties that are required to evaluate the body of the iterator-expression exp; we have organized accordingly the expressions in three groups: A, B, and C. Consider, for example, the evaluation

[5] In the case of MDT OCL, the scenario MyLibrary was loaded from an XMI file; for EOS, it was built using a Java program that simply calls the appropriate EOS's interface methods for defining the scenario; and for USE, it was loaded from a file that contained the appropriate USE commands to build the scenario.

[6] A more "natural" approach will be to consider scenarios with larger number of books but, as we will discuss in Section 4, none of the tools support well the loading of scenarios with more than 10^6 objects.

[7] We have also run the same experiments using a desktop computer, with Window Vista Business installed, a processor Intel Core 2 Quad CPU Q9300 2.50GHz 2.50 GHz, and 2GB of RAM. Again, in the case of the EOS and USE evaluators, we run the JVM with its parameters `-Xms` and `-Xmx` set to 1024m. In the case of EOS and MDT OCL the results were similar to those shown in Table 2. In the case of USE, however, the evaluations were completed, approximately, in half the time taken by the single-core test machine.

		MDT	EOS	USE
p_1		< 1s	30ms	1s12ms
p_2	->size()	< 1s	190ms	7s330ms
p_3		≈ 5s	931ms	1m22s
Group A		MDT	EOS	USE
p_1		< 1s	80ms	1s212ms
p_2	->collect(x|x.title)->size()	≈ 1s	391ms	10s84ms
p_3		≈ 18s	3s 896ms	1m48s
p_1		< 1s	90ms	981ms
p_2	->collect(x|x.title <> 'Hobbit')->size()	≈ 2s	481ms	8s432ms
p_3		≈ 20s	4s 516ms	1m30s
Group B		MDT	EOS	USE
p_1		< 1s	240ms	6s810ms
p_2	->collect(x|x.author.books)->size()	≈ 6s	2s 140ms	1m42s
p_3		≈ 58s	17s 736ms	*Error*
p_1		< 1s	221ms	3s565ms
p_2	->collect(x|x.author.books->includes(x))->size()	≈ 7s	1s 893ms	32s677ms
p_3		≈ 1m 1s	17s 906ms	5m32s
p_1		< 1s	251ms	3s475ms
p_2	->forAll(x|x.author.books->includes(x))	≈ 5s	1s 963ms	32s6ms
p_3		≈ 53s	17s 30ms	5m25s
p_1		< 1s	260ms	3s685ms
p_2	->select(x|x.author.books->includes(x))->size()	≈ 5s	2s 13ms	35s411ms
p_3		≈ 55s	17s 605ms	5m51s
Group C		MDT	EOS	USE
p_1		≈ 2s	290ms	8s412ms
p_2	->collect(x|x.author.books.title)->size()	≈ 20s	2s 573ms	1m27s
p_3		≈ 3m 17s	23s 684ms	*Error*
p_1		≈ 2s	270ms	4s957ms
p_2	->collect(x|x.author.books.title->size())->sum()	≈ 19s	2s 274ms	48s660ms
p_3		≈ 3m 15s	20s 840ms	10m54s
p_1		≈ 2s	280ms	4s777ms
p_2	->forAll(x|x.author.books.title->excludes('Hobbit'))	≈ 20s	2s 604ms	46s286ms
p_3		≈ 3m 15s	22s 802ms	7m52s

Table 2. Evaluating performance: MDT OCL, EOS, and USE.

times for the first expression in each group. Finally, Table 2 shows that, again for the same p_i, the time $exp(p_i)$ depends as well for these tools on the size of the collection that need to be built. Consider, for example, the evaluation time for the first two expressions in Group C: notice that to evaluate

$$p_3\text{−>collect(x|x.author.books.title)−>size()} \tag{3}$$

will require to allocate memory for storing a collection with 10^8 titles while evaluating

$$p_3\text{−>collect(x|x.author.books.title−>size())−>sum()} \tag{4}$$

will *only* require to allocate memory for storing a collection with 10^7 integers. Using these and similar expressions, we regularly use the above mentioned measurements, namely, the maximum number of times that objects' properties will be accessed and the maximum size of the collections that will be built, to check the performance of the EOS evaluator and look for possible optimizations.

3 The implementation of the EOS evaluator

As mentioned before, based on our executable equational semantics for OCL [11], we have implemented a rewriting-based OCL evaluator, named ITP/OCL [6]. Although our tool performs reasonably well on small-medium size scenarios, its performance does not scale up to medium-large size scenarios. Prompted by our interest on applications that require efficient OCL evaluation on medium-large size scenarios, we decided to implement the Eye OCL Software (EOS) evaluator, a Java component whose designed follows the key ideas behind the ITP/OCL tool.

The implementation of the EOS component has taken 4 man-months. It includes an OCL parser (which uses SableCC) and an OCL evaluator, the latter consisting of about 7K lines of Java code. Although its first public released is not expected until October 2008, the current beta version handles most of OCL, including the possibility of adding user-defined operations. With the idea of making it as 'pluggable' as possible, the EOS component is not based on any particular (meta)modeling framework: its public interface provides methods to insert elements, one-by-one, into user-models and scenarios, and to input the expressions to be evaluated as strings of ASCII characters. This decision allowed us also to design the EOS's data structure for internally storing user-models and scenarios in such a way that objects' properties are efficiently accessed. The other possible novelty in its implementation is that, before evaluating a collect expression, we try to (over)estimate the size of the resulting collection and allocate memory in advance. The rest of the EOS implementation is rather straightforward: OCL iterator-expressions are executed using Java for/while loops and standard OCL operations, when possible, are executed using the appropriate Java functions. Finally, expressions are evaluated in EOS following an eager strategy: in particular, collection-expressions are fully evaluated and their resulting elements are all allocated in memory. As part of our research agenda, we plan to study the advantages/disadvantages of a lazy strategy for OCL evaluation, where collection-expressions are only evaluated on-demand.

4 Dealing with really large scenarios

To evaluate expressions on really large scenarios, we need first to solve the problem of loading the scenarios in the OCL evaluators. To illustrate this challenge, we show in Table 3 the time and memory taken by MDT OCL, EOS, and USE, when loading different scenarios of the model Library. Each of the scenarios is identified by a number n, which also indicates its "size": more precisely, for each n, the scenario #n exactly contains 10^n books. Notice that for scenario #6, with 10^6 books, none of the tools were able to finish in less than 20 minutes.[8]

	MDT OCL			EOS		USE	
Scenario	XMI	Mem	Time	Mem	Time	Mem	Time
#3	50KB	111MB	< 1s	20MB	40ms	88MB	1s
#4	500KB	112MB	< 1s	22MB	661ms	116MB	35s
#5	5MB	125MB	≈ 45s	50MB	2m36s	209MB	8m25s
#6	50MB		> 20m		> 20m		> 20m

Table 3. Evaluating loading cost: MDT OCL, EOS, and USE.

So far, we have explored two different approaches for addressing this problem, both based on the representation of user-models and scenarios as relational databases. The first approach consists on modifying the EOS evaluator so as to look for the information contained in the scenarios directly in its database representation. The advantage of this approach is that it only requires modifying the evaluation of dot-expressions in the expected way: namely, accessing the value of an object's attribute or the value of an object's association-end will be now implemented as a basic SQL `select`-query. The concrete form of these queries depends, of course, on the mapping used to represent models as relational databases (see, for example, [21, 22] and [14]). To check the feasibility of this approach, we modified the EOS evaluator accordingly: unfortunately, the cost of evaluating dot-expressions through the JDBC driver was so high that it made impractical the use of the modified EOS evaluator for evaluating expressions on medium-large scenarios.[9]

[8] The "extra" time taken by the EOS tool to store scenarios (with respect at least to MDT OCL) is possibly due to the extra computation required to store scenarios in the EOS internal data structure.

[9] For this experiment, we mapped each class to a table whose columns correspond to its attributes, and each association to a table whose columns correspond to its association-ends.

The second approach consists on translating OCL expressions into expressions in a query language already available for relational databases. To the best of our knowledge, the most interesting results in this line are discussed in [9, 18] and provide the foundations of the OCL2SQL tool [17, 16]. However, the solution offered in [9, 18] is not satisfactory yet. First, it only considers a restricted subset of the OCL language: in particular, it cannot deal with tuples or nested collections. Second, it only applies to boolean expressions and not to arbitrary queries. Finally, the "complexity" of the SQL expressions resulting from this translation is so high that makes also impractical its use for evaluating expressions on medium-large scenarios.[10] For example, consider a simple extension of model Library shown in Figure 1 where books have now an additional attribute pages. Then, consider the following invariant:

$$\textbf{context}\ \mathsf{Writer}\ \textbf{inv}\text{: } \mathsf{self.books{-}{>}forAll(x \mid x.pages > 300)} \qquad (5)$$

Applying to (5) the translation implemented in OCL2SQL, we automatically obtain the SQL query shown in Figure 2. In Table 4 we show the results of evaluating this query on two different scenarios: scenario #1 contains 10^2 writers and 10^4 books, each writer being the author of 10^2 different books; and scenario #2 contains 10^2 writers and 10^5 books, each writer being the author of 10^3 different books. In both scenarios, all books have exactly 150 pages. The figures correspond to the local execution of the above query in a PostgreSQL 8.3 database installed in a laptop computer, with Windows XP Professional, a processor Intel Pentium M 2.00GHz 600MHz, and 1GB of RAM.

Scenario	Time
# 1	≈ 0m25s
# 2	≈ 45m

Table 4. Evaluating performance: OCL2SQL

5 Conclusions

In this paper we have first motivated the need for efficient OCL evaluation support in order to cope with the high-computational cost of evaluating OCL expressions in medium-large size scenarios. Then, we have discussed, based on a number of experiments, two measurements that should be taken into consideration when building OCL evaluators for medium-large scenarios: namely, the cost of accessing individual objects' properties and the cost of building collections to hold the partial results of an evaluation. Independently of the results

[10] The OCL2SQL's main developer has confirmed that the efficiency of the OCL2SQL tool has not been tested on medium-large scenarios (e-mail communication, May 2008).

```
create or replace view NAME  as
  (select *
   from OV_Writer as SELF
   where not (not exists (
         select PK_Book
         from (select PK_Book
               from OV_Book as foo
               where PK_Book in
                     (select FK_books
                      from ASS_Ownership as foo
                      where FK_author in
                            (select PK_Writer
                             from OV_Writer as foo
                             where PK_Writer = SELF.PK_Writer)))
        as foo
        where PK_Book in (
              select PK_Book
              from OV_Book as ALIAS2
              where not (ALIAS2.pages> 300)
        )
        )))
```

Fig. 2. An example of an SQL query generated by OCL2SQL.

of our benchmark, our aim here is similar to that of [12]: we do not want to recommend the use of a particular tool, but would like to emphasize the need for a benchmark for OCL engine efficiency on *medium-large* scenarios which can help to build OCL implementations. Next, we have briefly discussed the implementation of the EOS evaluator [8]. Finally, we have presented the challenge of evaluating expressions on really large scenarios and discussed the feasibility of various approaches to address this problem.

References

1. T. Baar and S. Markovic. The RoclET tool. `http://www.roclet.org/index.php`, 2007.
2. A. Baroni and F. Brito e Abreu. An OCL-based formalization of the MOOSE metric suite. In *ECOOP Workshop on Quantitative Approaches in Object Oriented Software Engineering*, Darmstadt, Germany, July 2003.
3. A. L. Baroni and F. Brito e Abreu. A formal library for aiding metrics extraction. In *International Workshop on Object-Oriented Re-Engineering at ECOOP'2003*, Darmstadt, Germany, July 2003.
4. D. Basin, M.Clavel, J. Doser, and M. Egea. Automated analysis of security-design models. *Information and Software Technology*, 4853, 2008. To appear in the special issue on *Model Based Development for Secure Information Systems*.
5. D. Chiorean, M. Bortes, D. Corutiu, C. Botiza, and A. Carcu. An OCL environment (OCLE) 2.0.4. `http://lci.cs.ubbcluj.ro/ocle/`, 2005. Laboratorul de Cercetare in Informatica, University of BABES-BOLYAI.

6. M. Clavel and M. Egea. ITP/OCL: A rewriting-based validation tool for UML+OCL static class diagrams. In *AMAST'06: 11th International Conference on Algebraic Methodology and Software Technology*, volume 4019 of *LNCS*, Kuressaare, Estonia, July 2006. Springer-Verlag.
7. M. Clavel, M. Egea, and V. Torres. Model metrication in MOVA: A metamodel based approach using OCL. `http://maude.sip.ucm.es/~marina/pubs/pubs.html`, 2007.
8. M. A. García de Dios, M. Clavel, and M. Egea. The Eye OCL Software (EOS), 2008. `http://maude.sip.ucm.es/eos`.
9. B. Demuth, H. Hussmann, and S. Loecher. OCL as a specification language for business rules in database applications. In M. Gogolla and C. Kobryn, editors, *UML 2001: 4th International Conference on the Unified Modeling Language*, volume 2185 of *LNCS*, Toronto, Canada, 2001. Springer.
10. F. Brito e Abreu. Using OCL to formalize object oriented metrics definitions. Technical Report ES007/2001, FCT/UNL and INESC, Portugal, June 2001. `http://ctp.di.fct.unl.pt/QUASAR/Resources/Papers/others/MOOD_OCL.pdf`.
11. M. Egea. *An executable formal semantics for OCL with Applications to Formal Analysis and Validation*. PhD thesis, Universidad Complutense de Madrid, 2008. `http://maude.sip.ucm.es/~marina/pubs/`. Submitted.
12. M. Gogolla, M. Kuhlmann, and F. Büttner. A benchmark for OCL engine accuracy, determinateness, and efficiency. To appear in *Proceedings of ACM/IEEE 11th International Conference on Model Driven Engineering Languages and Systems (28 September-3 October, 2008, Toulouse, France)*, 2008.
13. M. Gogolla, M. Kuhlmann, and F. Büttner. Sources for a benchmark for OCL engine accuracy, determinateness, and efficiency. `http://www.db.informatik.uni-bremen.de/publications/Gogolla_2008_BMSOURCES.pdf/`, 2008.
14. D. Gornik. A UML data modeling profile. `http://www.jeckle.de/files/RationalUML-RDB-Profile.pdf`, February 2005. IBM.
15. Database Systems Group. The UML specification environment (USE) tool, 2006. `http://www.db.informatik.uni-bremen.de/projects/USE/`.
16. Software Technology Group. The OCL2 Dresden toolkit. `http://sourceforge.net/project/showfiles.php?group_id=5840`, 2007.
17. F. Heidenreich. OCL-codegenerierung für deklarative Sprachen. Master's thesis, University of Dresden, March 2006. `http://dresden-ocl.sourceforge.net/publications.html`.
18. F. Heidenreich, C. Wende, and B. Demuth. A framework for generating query language code from OCL invariants. In *7th OCL Workshop at the UML/MoDELS Conference*, 2007.
19. Kenn Hussey and Model Development Tools Team. MDT-OCL. `http://www.eclipse.org/modeling/mdt/?project=ocl`, 2008. Inside Eclipse Ganymede Release.
20. Object Management Group. *Object Constraint Language specification*, May 2006. OMG document available at `http://www.omg.org/docs/ptc/05-06-06.pdf`.
21. S. Sambasivam and P. Crefcoeur. How databases and XML can be used to master UML models, an investigation. *J. Comput. Small Coll.*, 23(6):220–228, 2008.
22. Y. Shuxin and R. Indrakshi. Relational database operations modeling with UML. In *AINA '05: Proceedings of the 19th International Conference on Advanced Information Networking and Applications*, pages 927–932, Washington, DC, USA, 2005. IEEE Computer Society.