2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)

A Data Set of OCL Expressions on GitHub

Jeroen Noten, Josh G.M. Mengerink, Alexander Serebrenik
Eindhoven University of Technology
The Netherlands
Email: j.f.h.noten@student.tue.nl, {j.g.m.mengerink, a.serebrenik}@tue.nl

***Abstract*—In model driven engineering (MDE), meta-models are the central artifacts. As a complement, the Object Constraint Language (OCL) is a language used to express constraints and operations on meta-models. The Eclipse Modeling Framework (EMF) provides an implementation of OCL, enabling OCL annotated meta-models.**

Existing empirical studies of the OCL have been conducted on small collections of data. To facilitate empirical research into the OCL on a larger scale, we present the first publicly-available data set of OCL expressions. The data set contains 9188 OCL expressions originating from 504 EMF meta-models in 245 systematically selected GitHub repositories. Both the original meta-models and the generated abstract syntax trees are included, allowing for a variety of empirical studies of the OCL. To illustrate the applicability of this data set in practice, we performed three case studies.

***Keywords*-OCL, GitHub, data set.**

I. Introduction

Model driven engineering (MDE) is being used in industry to drive increase in productivity [12], e.g., using domain specific languages (DSLs). DSLs are often based on meta-models. Meta-models define the concepts and structure of the domain models that can be built using a DSL. When designing complex DSLs, the expressivity of meta-models alone is often not sufficient to accurately specify the domain [20]. To address this problem, more complex mechanisms have been proposed, such as the Object Constraint Language (OCL) [22].

While empirical studies of domain specific languages and meta-models have been conducted in the past [9], [10], [11], [16], [18], [23], little attention of the research community has been spent on empirical evaluation of OCL [5], [6], [19]. Reynoso et al. [19] and Correa et al. [6] conducted controlled experiments on specially prepared models as opposed to the real-world ones, and Cadavid et al. [5] have studied a relatively small collection of 840 OCL expressions derived from 34 publicly available and 3 commercial meta-models.

To facilitate further empirical studies of OCL we present a data set of 9188 OCL expressions derived from 504 meta-models available on GitHub. Availability of this data set will enable the researchers

- to replicate earlier studies such as the one by Cadavid et al. [5] on larger and more diverse data sets;
- to replicate corpus-based studies conducted in the context of traditional software engineering, in a MDE context;
- to provide a complementary perspective on the earlier results obtained through controlled experiments [6], [19];
- to evaluate practical limitations of techniques proposed to analyze [2], [15] and visualize OCL [4];
- to compare characteristics of the OCL expressions from open source projects with the previously published characteristics of the limited number of closed-source projects (cf. [5]), validating open source OCL as a vehicle for further studies.

Furthermore, it will allow the companies specializing in software quality to benchmark OCL expressions from systems under investigation against a larger collection of OCL expressions, and thus derive conclusions about the system quality (cf. similar work for non-MDE software [8], [17]).

The remainder of this paper is organized as follows. First we explain how we collected the data for our data set. Next we provide a description of the data set. Then, we demonstrate three case studies with our data set. Finally, we explain the limitations and then conclude this paper.

II. GitHub Data collection

A. GitHub as platform for OCL data collection

Several collections of open-source MDE projects are publicly available. For OCL, Jordi Cabot has compiled the OCL repository[1]. However, this collection is relatively small (105 `.ocl` files and 2 `.ecore` files). Also, it does not have a clear structure, which hinders automated analysis. Other examples of meta-model collections include MDE Forge[2] and ReMoDD[3]. However, none of the meta-models at MDE Forge contain any OCL expression, and the ReMoDD collection contains only 81 meta-modeling related artifacts.

To create a more representative, and up-to-date data set we mine public GitHub repositories. We chose GitHub, as it is the largest source of open-source in-development software systems. Moreover, previous studies into the usage of modeling-related technologies [7], [14] have also used the GitHub data. Finally, by focussing on GitHub we ensure that our data set includes the aforementioned OCL repository of Jordi Cabot.

OCL expressions can be stored either in a separate file or as part of a file containing the meta-model to which the OCL code refers. The naming convention of Eclipse, the most active open-source MDE community [14], requires extensions `.ocl` for OCL-only files and `.ecore` for meta-models.

[1] https://github.com/jcabot/ocl-repository
[2] http://www.mdeforge.org
[3] http://www.cs.colostate.edu/remodd/v1

978-1-5386-1544-7/17 $31.00 © 2017 IEEE
DOI 10.1109/MSR.2017.52

B. GitHub search

GitHub provides advanced search features, allowing one to look for different artifacts (e.g., commits, code files or wiki entries) containing or not containing given search strings, created during a given time period and owned by given users. Intuitively, we would like to search GitHub for all `.ocl` files, and `.ecore` files containing OCL code.

However, GitHub requires at least one search term to be included *in addition to* the requirement that a file has a given extension, i.e., one cannot identify every file with a particular extension. Kolovos et al. [14] also faced this limitation. To mitigate this problem the authors constructed search terms that provided the largest number of relevant results. For `.ocl` they, e.g., included the term *context*. However, this query does not match `.ocl` files without the term *context*. Using the query *extension:ocl NOT context*, we conclude that 999 code results would have been missed using the method of Kolovos et al.

Therefore, we suggest an alternative approach to query construction. As the search string we take *the negation of a search term that matches no `.ocl` files*. This negated term, by definition, matches all `.ocl` files. In our case the search term "foofoo" returned no results for files with the `.ocl` extension. (i.e., the query *extension:ocl foofoo* returned no results). Hence, the query *extension:ocl NOT foofoo* yields all files with the `.ocl` extension.

Next, we create a query that matches `.ecore` files containing OCL constraints. To enable the use of embedded OCL constraints in a `.ecore` file, the `.ecore` file should contain an annotation with the value "http://www.eclipse.org/emf/2002/Ecore/OCL"[4]. This value is persisted verbatim, and we use it to search for `.ecore` files containing OCL.

Hence, we look for the "code" results of the following queries: (1) *extension:ocl NOT foofoo*; (2) *extension:ecore "http://www.eclipse.org/emf/2002/Ecore/OCL"*. In March 2017 the queries produced 6237 and 1045 hits, respectively. As an `.ocl` file requires one or more corresponding `.ecore` files to be parsed. To ensure that all our data is processable, we need to also obtain these `.ecore` files. On GitHub, every *code match* belongs to a file in a repository. Rather than identifying single files, we download the full repository of the files identified by our queries, and identify the files required for parsing OCL offline.

The next limitation of GitHub search is that only 1000 results are retrieved. We circumvent this by incrementally modifying our search query: we exclude repositories that we have already been found in previous iterations (using *-repo:[user]/[repo]*). For example, if in the first iteration we find only code results from the repositories *eclipse/ocl* and *eclipse/ecore*, the query for our next iteration will be *extension:ecore http://www.eclipse.org/emf/2002/Ecore/OCL -repo:eclipse/ocl -repo:eclipse/ecore*. We repeat this procedure as long as results are retrieved. Finally, all excluded repositories form the list of relevant repositories.

[4]http://help.eclipse.org

The last limitation that we encountered during our search process is the limitation of the search query length. However, this did not lead to problems, because when the query length reached this limit, the number of code results was less than 1000 (the maximum number of search results shown by GitHub), so instead of excluding more repositories from the search, all repositories that occurred in the results are added to the list of relevant repositories.

As a result of this search process, we have two lists of repositories, one for each of the two search queries. Merging these lists and eliminating duplicates yields a list of 519 relevant repositories.

C. Downloading and stripping the repositories

Using a Python script, we download the 519 relevant repositories. Next, we remove all files and empty directories other than files ending in `.ocl` and `.ecore`, as we only want to keep those for our data set. This results in a collection of of 6258 `.ocl` files and 21188 `.ecore` files. In order to keep the files parsable we preserve the original file names and the directory structure.

D. Parsing

We have observed that there are many duplicate files both in the same repository as well as across repositories. This happens for example when files or directories are copied or when dependencies are included. To prevent bias in the usage statistics, we only want to include unique files.

Hence, we first identify duplicates using MD5 hashing of the files (i.e., their content). In the collection of 27446 (6258 + 21188) files, only 10894 files are unique (spread over the 519 repositories).

Using the Eclipse Modeling Framework, we parse all unique files and store them as abstract syntax trees (ASTs) in XMI format conforming to the OCL Pivot Meta Model [24]. We successfully parse 8947 files (76%) resulting in 8947 AST files. The remaining 2759 files resulted in parse errors, due to e.g., the extension `.ocl` being used for technologies not related to the Object Constraint Language, missing references, or syntax errors. 274 of the 519 repositories contained no parsable OCL constraints at all. Since we are only interested in files with (parsable) OCL expressions, we exclude AST files with no parsable OCL expressions. This step resulted 504 AST files containing 9188 OCL expressions, derived from 245 $(519 - 274)$ repositories.

III. Data description

The dataset, as well as the scripts that we wrote to compose it, is publicly available on GitHub[5].

The `repos` directory contains data from the 245 repositories identified in the previous section. Directories imitate the structure of the repositories; however, all files other than `.ocl` and `.ecore` are removed, as well as directories that are empty as a result. For each repository, a `.json` file with metadata is included.

[5]https://github.com/tue-mdse/ocl-dataset

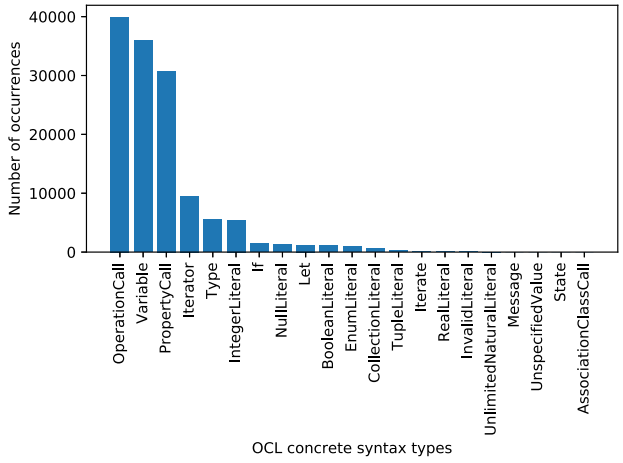

Fig. 1. Frequency of OCL constructs in analyzed expressions.

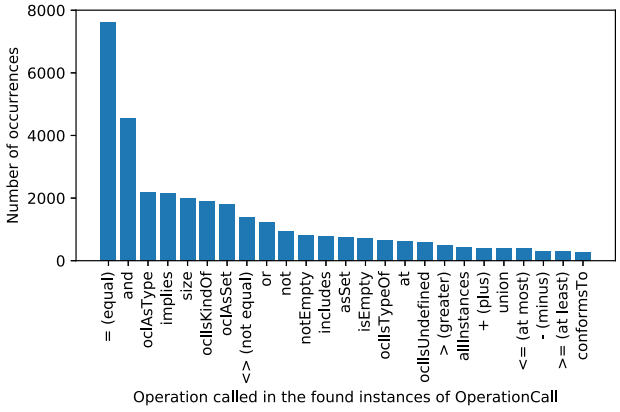

Fig. 2. Frequency of top 25 of operations called in occurrences of `OperationCall`.

The `oclas` directory contains files with the abstract syntax trees (ASTs) of the `.ocl` and `.ecore` files. The ASTs are stored in XMI format conforming to the OCL Pivot Meta-Model [24]. The files are named by the MD5 hashes of their contents, and have the `.oclas` extension. We identify the files using MD5 since in Section II-D we have eliminated multiple occurrences of files with the identical contents.

The `meta.json` file contains meta information about the repositories and the AST files. The JSON object in this file contains two keys: `oclas` and `repos`. The `oclas` object maps the names of the AST files to an array of `.ecore` or `.ocl` files (of which the contents are identical) that the AST refers to. The `repos` object maps the names of the repositories to commit hashes that were the most recent commits when we downloaded the repositories.

IV. Applications

In the introduction we mentioned several possible applications of this data set. To show how our data set can be used in practice, we (partially) replicate the study of Cadavid et al. [5] and discuss the practicality of the assumptions made by Anastakis et al. [2].

Cadavid et al. [5] analyzed the practical usage of OCL in 37 meta-models. To perform a partial replication of this study, we reuse the original methodology and apply it to our data set of OCL expressions in 504 meta-models. According to the definition of Shull et al. [21], we perform an exact replication.

A. Measuring the frequency of OCL constructs

Using the EMF, we count the frequency of each OCL construct in all AST files. Figure 1 shows these frequencies.

These results are in line with the original results by Cadavid et al. [5], e.g., the set of the 6 of most used constructs is the same. Also, the chart looks similar, however we cannot measure the differences precisely due to the lack of exact numbers in the original study. They present a percentage that we can compare though: they name 10 constructs that capture 98.6% of all constructs present. Calculating this statistic using our data set gives 97.8%. Hence the original result is reproducible, which indicates a gain in confidence of the original study.

B. Measuring the frequency of called operations

We also reproduced another metric from the same paper: the frequency of operations called in occurrences of `OperationCall` (the most used OCL construct). Figure 2 shows the resulting frequencies. We only show the top 25 operations, to make the chart readable.

In contrast to the previous one, this chart is *not* in line with the original results. We mention a few notable differences: the most used operation in the original paper is not even in our top 25. Furthermore, our second most used operation 'and' is at place 21 in the original paper. These and other notable differences imply that the scope of the conclusions from this data in the original paper is limited.

C. Limiting a threat to validity of another study

In our third case study, we investigate a threat to validity of another paper. Anastasakis et al., study the difference in expressivity between OCL and Alloy [2].

They conclude that there are various OCL concepts that cannot be expressed in Alloy, possibly rendering their technique inapplicable in practice. Their leading example is the `Iterate` operation, which has no equivalent in Alloy. In our data set, the `Iterate` operation is used in 8% $(19/245)$ of the projects. More specifically 5% $(24/504)$ of the metamodels in our dataset use this particular OCL constraint. However, in the metamodels, only 1% $(82/9188)$ of the OCL expressions incorporate this operation. Hence, most OCL expressions are not affected, and the technique of Anastakis et al. can be applied to almost all expressions.

V. Limitations

As with any empirical research, the data collection process is subject to several threats to validity.

Several threats to validity have been introduced by our decision to *use GitHub*. The "peril of mining GitHub" [13] most relevant for our work is that "many active projects do not conduct all their software development on GitHub". To mitigate this threat, as a future work we plan to extend the data set with additional sources of data, such as SourceForge, OMG documents, and scientific articles.

The limitations of the *search functionality of GitHub* [1] also induce several threats to validity. The search functionality of GitHub only allows searching of the main branch in repositories, i.e., our search query might miss files [3]. However, our data is less likely to contain experimental files, giving a more accurate representation of finished products. Similarly, files might be missed due to project forks being excluded by default from the GitHub search. While, in general, this is beneficial as it reduces noise in the data, it is also possible that forks contain new data as well, which we then miss.

Moreover, only files smaller than 384 KB are searchable. This means that the search misses repositories in which *all* `.ocl` and `.ecore` files exceed 384KB (note that we download full repositories that we identified, potentially including files bigger than 384KB). To estimate the number of `.ecore` and `.ocl` files larger than 384KB, we investigate the repositories that we included in the data set. We conclude that of all `.ecore` and `.ocl` files in the repositories, 3% (739/25130) is bigger than 384 KB. We therefore expect the impact of this threat to be limited.

Finally, another limitation of the search pertains to very big repositories: GitHub search covers only repositories with fewer than 500,000 files. This may cause us to miss files in very large repositories.

VI. Conclusions

In order to facilitate empirical research into the OCL, we presented a data set of 9188 OCL expressions, derived from 504 unique `.ocl` and `.ecore` files, originating from 245 systematically selected GitHub repositories. The data set includes the original `.ocl` and `.ecore` files, as well as the generated AST files. The AST files are stored in XMI format conforming to the OCL Pivot Meta Model.

This data set allows for a variety of empirical studies of the OCL, including usage studies and practical evaluations of proposed techniques. We performed three case studies to illustrate the applicability of this data set in practice. Two of these case studies are replications of a usage study on a smaller scale. In the third one, we use our data set to limit a thread to validity of a previous study.

As future work, we consider extending our data set with OCL expressions and corresponding meta-models from other sources such as Google Code, SourceForge and data sources used by existing research into the OCL.

References

[1] GitHub help — searching code. https://help.github.com/articles/searching-code/. Accessed: 2017-03-14.

[2] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. On challenges of model transformation from uml to alloy. *Software & Systems Modeling*, 9(1):69, 2008.

[3] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. Germán, and Premkumar T. Devanbu. The promises and perils of mining git. In *MSR*, pages 1–10, 2009.

[4] Paolo Bottoni, Manuel Koch, Francesco Parisi-Presicce, and Gabriele Taentzer. *A Visualization of OCL Using Collaborations*, pages 257–271. Springer, 2001.

[5] Juan José Cadavid, Benoit Combemale, and Benoit Baudry. An analysis of metamodeling practices for MOF and OCL. *Computer Languages, Systems & Structures*, 41:42–65, 2015.

[6] Alexandre Correa, Cláudia Werner, and Márcio Barros. *An Empirical Study of the Impact of OCL Smells and Refactorings on the Understandability of OCL Specifications*, pages 76–90. Springer, 2007.

[7] Regina Hebig, Truong Ho Quang, Michel RV Chaudron, Gregorio Robles, and Miguel Angel Fernandez. The quest for open source projects that use UML: mining GitHub. In *MODELS*, pages 173–183. ACM, 2016.

[8] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In Ricardo Jorge Machado, Fernando Brito e Abreu, and Paulo Rupino da Cunha, editors, *QUATIC*, pages 30–39. IEEE, 2007.

[9] Felienne Hermans, Martin Pinzger, and Arie van Deursen. *Domain-Specific Languages in Practice: A User Study on the Success Factors*, pages 423–437. Springer, 2009.

[10] John Hutchinson, Mark Rouncefield, and Jon Whittle. Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 633–642. ACM, 2011.

[11] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of mde in industry. In *ICSE*, pages 471–480. IEEE, 2011.

[12] John Edward Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of MDE in industry. In *ICSE*, pages 471–480, 2011.

[13] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela Damian. The promises and perils of mining GitHub. In *MSR*, pages 92–101, 2014.

[14] Dimitrios S Kolovos, Nicholas Drivalos Matragkas, Ioannis Korkontzelos, Sophia Ananiadou, and Richard F Paige. Assessing the use of eclipse mde technologies in open-source software projects. In *OSS4MDE@ MoDELS*, pages 20–29, 2015.

[15] Mirco Kuhlmann, Lars Hamann, and Martin Gogolla. Extensive validation of OCL models by integrating SAT solving into USE. In Judith Bishop and Antonio Vallecillo, editors, *TOOLS*, pages 290–306. Springer, 2011.

[16] Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, and Miguel A. Fernandez. An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. *Empirical Software Engineering*, 18(1):89–116, 2013.

[17] Paloma Oliveira, Fernando Paim Lima, Marco Tulio Valente, and Alexander Serebrenik. RTTool: A tool for extracting relative thresholds for source code metrics. In *ICSME*, pages 629–632. IEEE, 2014.

[18] Marian Petre. Uml in practice. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 722–731. IEEE Press, 2013.

[19] Luis Reynoso, Marcela Genero, Mario Piattini, and Esperanza Manso. Does object coupling really affect the understanding and modifying of OCL expressions? In *SAC*, pages 1721–1727. ACM, 2006.

[20] Mark Richters and Martin Gogolla. On formalizing the UML object constraint language OCL. In *Conceptual Modeling*, pages 449–464, 1998.

[21] Forrest J Shull, Jeffrey C Carver, Sira Vegas, and Natalia Juristo. The role of replications in empirical software engineering. *Empirical Software Engineering*, 13(2):211–218, 2008.

[22] Jos Warmer and Anneke Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. Addison-Wesley, 2 edition, 2003.

[23] Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE software*, 31(3):79–85, 2014.

[24] Edward D. Willink. Aligning OCL with UML. *ECEASST*, 44, 2011.