Empir Software Eng (2019) 24:1574–1609
https://doi.org/10.1007/s10664-018-9641-6

Empowering OCL research: a large-scale corpus of open-source data from GitHub

Josh G. M. Mengerink[1] · **Jeroen Noten**[1] · **Alexander Serebrenik**[1]

Published online: 23 August 2018

Abstract Model-driven engineering (MDE) enables the rise in abstraction during development in software and system design. In particular, meta-models become a central artifact in the process, and are supported by various other artifacts such as editors and transformation. In order to define constraints, invariants, and queries on model-driven artifacts, a generic language has been developed: the Object Constraint Language (OCL). In literature, many studies into OCL have been performed on small collections of data, mostly originating from a single source (e.g., OMG standards). As such, generalization of results beyond the data studied is often mentioned as a threat to validity. Creation of a benchmark dataset has already been identified as a key enabler to address the generalization threat. To facilitate further empirical studies in the field of OCL, we present the first large-scale dataset of 103262 OCL expression, systematically extracted from 671 GitHub repositories. In particular, our dataset has extracted these expressions from various types of files (a.o. metamodels and model-to-text transformations). In this work we showcase a variety of different studies performed using our dataset, and describe several other types that could be performed. We extend previous work with data and experiments regarding OCL in model-to-text (mtl) transformations.

Keywords Dataset · Model-driven engineering · OCL · EMF · Acceleo · Replication studies

Communicated by: Abram Hindle and Lin Tan

✉ Alexander Serebrenik
a.serebrenik@tue.nl

Josh G. M. Mengerink
j.g.m.mengerink@tue.nl

1 Eindhoven University of Technology, Eindhoven, The Netherlands

1 Introduction

Model driven engineering (MDE) is being used in industry to drive increase in productivity (Hutchinson et al. 2011b). One such driver is the use of domain specific languages (DSLs) to allow engineers to specify systems in terms relevant to their domain, rather than encoding them into general purpose concepts like those of UML. These DSLs are underpinned by *metamodels* (Cuadrado and Molina 2007), which express the concepts and structure of possible models (i.e., abstract syntax). However, as DSLs grow in complexity, the expressivity of meta-models alone is often not sufficient to accurately specify the domain (Richters and Gogolla 1998). To address this problem, more complex mechanisms have been proposed, such as the Object Constraint Language (OCL) (Warmer and Kleppe 2003).

While empirical studies of domain specific languages and meta-models have been conducted in the past (Hermans et al. 2009; Hutchinson et al. 2011a, b; Mohagheghi et al. 2013; Petre 2013; Whittle et al. 2014), relatively little attention of the research community has been spent on empirical evaluation of OCL (Cadavid et al. 2015; Correa et al. 2007; Reynoso et al. 2006). A core reason for this is that *OCL data is scarce*. So far empirical studies have been solely conducted either on synthetic models (rather than real-world ones) or on smaller datasets: Reynoso et al. (2006) and Correa et al. (2007) conducted controlled experiments on synthetic models, and Cadavid et al. (2015) have studied a relatively small collection of 840 OCL expressions derived from 34 publicly available and three commercial meta-models. Attempts at creating larger datasets have been made by Cabot,[1] and Basciani et al. (2014). Despite these efforts, a need for a more diverse and thorough dataset has been recognized in the literature (Gogolla et al. 2013; Gogolla and Cabot 2016).

This work builds on our previously published dataset (Noten et al. 2017a) of 9173 OCL expressions derived from 504 open-source meta-models, and earlier experimentation by Mengerink et al. (2017a). We build upon both works by extending the dataset with model-to-text transformations and feature several experiments that use the 94089 expressions derived from 2634 open-source model-to-text transformations available on GitHub. The extended dataset, in addition to the scripts written to compose it, is publicly available on GitHub.[2] To ensure persistence and permanent identification of the data we have registered it with the 4TU federation through DataCite Netherlands (Noten et al. 2017b). We show that, using the extended dataset, one is able to:

1. replicate earlier studies such as the one by Cadavid et al. (2015) on larger and more diverse data sets;
2. evaluate practical limitations of techniques proposed to analyze (Anastasakis et al. 2008; Kuhlmann et al. 2011) and visualize OCL (Bottoni et al. 2001);
3. compare characteristics of the OCL expressions from open source projects with the previously published characteristics of the limited number of closed-source projects (cf. Cadavid et al. 2015, Mengerink et al. 2017a, b), validating open source OCL as a vehicle for further studies.

Moreover, we expect the dataset to be applicable in other ways, e.g., to:

1. provide a complementary perspective on the earlier results obtained through controlled experiments (Correa et al. 2007; Reynoso et al. 2006);

[1] https://github.com/jcabot/ocl-repository

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

2. replicate corpus-based studies conducted in the context of traditional software engineering, in a MDE context;

Our dataset (Noten et al. 2017b) will also allow 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 Heitlager et al. 2007; Oliveira et al. 2014).

The remainder of this paper is organized as follows. After introducing MDE and OCL in Section 2, in Section 3, we discuss how our dataset was composed, followed by the dataset description in Section 4. In Section 5 we illustrate applications of the dataset by showing several studies that can be performed. Next we review the related work in Section 6 and possible threats to validity in Section 7. Finally, we conclude with discussing future work and summarizing our contribution in Section 8.

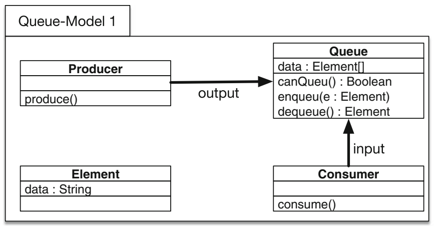
2 A Brief Introduction to MDE and OCL

2.1 Model-Driven Engineering

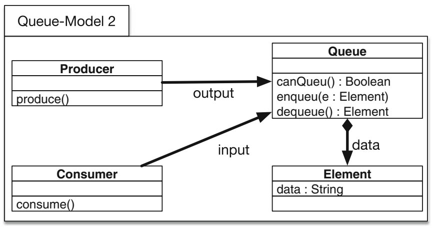
Model-driven engineering (MDE) (Kleppe et al. 2003) puts models central in the development process. Based on such a model, engineers can perform simulation and verification early in the design process. Based on this early verifications, designs can be adapted, even before implementation. When satisfied that a model describes the system as desired, generators allow the automatic creation of traditional design artifacts from these models. This way of working raises the level of abstraction, relieving the engineer from thinking about implementation issues. For instance, an engineer can specify that a queue should be present whilst refraining from deciding what type of mutual-exclusion protocol should be used for that queue.

2.2 Domain Specific Language

However, using MDE still does not fully exploit the raise of abstraction: the engineer still has to encode their specific problem into a generic modeling language (e.g., UML 1997 or SysML 2001). As such, the engineer is still faced with unnecessary design decisions (e.g., encoding their `Element` as `Class` or as a `DataType` as illustrated in Fig. 1). To mitigate this, domain-specific languages (DSLs) aim at enabling specification of a problem in terms



(a) A queue-pattern where the data is a separate datatype



(b) A queue-pattern where the data is a child of the queue

Fig. 1 Two different ways to encode a producer-queue-consumer in a general purpose modeling language

of relevance to the engineers domain. For instance, not having the engineer reason in terms of classes and references, but in terms of queues, producers, and consumers.

2.3 (Meta)modeling

A way to easily create domain-specific languages using MDE is through the use of metamodels. Metamodels describe, for a DSL, what constitutes a valid model. In essence, they are a model of all models (e.g., a metamodel) contained "in the language". The Eclipse Modeling Framework (EMF) is a commonly used framework for creating metamodels. It uses (Ecore 2004) as central formalism, which implements the Meta-Object Facility (MOF 1996) standard described by the object management group (OMG 1989). Ecore, in turn, describes what constitutes a valid metamodel. Being a model of metamodels, it is what is referred to as a meta-metamodel. A fragment of the Ecore meta-metamodel is illustrated in Fig. 2.

As an example, consider a simple DSL describing the movement of a robot, as illustrated in Fig. 4a. The movement of this robot is described in a `Specification`, which is comprised of a sequence of `Commands`. Command, as an abstract class, cannot be instantiated itself, but its two subclasses can. There, `Forward` represents telling the robot to move forward, and `Turn` represents telling the robot to make a 90° clockwise turn. The metamodel for our example DSL is illustrated in Fig. 4a. As the relation between such a metamodel (in its concrete syntax) and the meta-metamodel in Fig. 2 are hard to read, we provide the same metamodel, in abstract syntax, in Fig. 3.

Using the metamodel and the infrastructure EMF provides for it, we are now able to specify models, which are "instances of" (cf. Bézivin 2006) their metamodel. An example model for the metamodel in Figs. 3 and 4a is illustrated in Fig. 4b. This model corresponds to the robot moving forward, turning 90° clockwise and moving forward again.

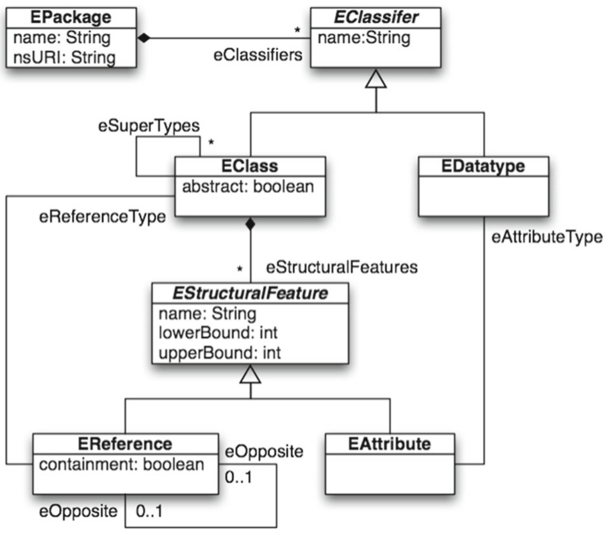


Fig. 2 A fragment of the Ecore meta-meta-model taken from the Eclipse Juno documentation (Eclipse Juno 2012)

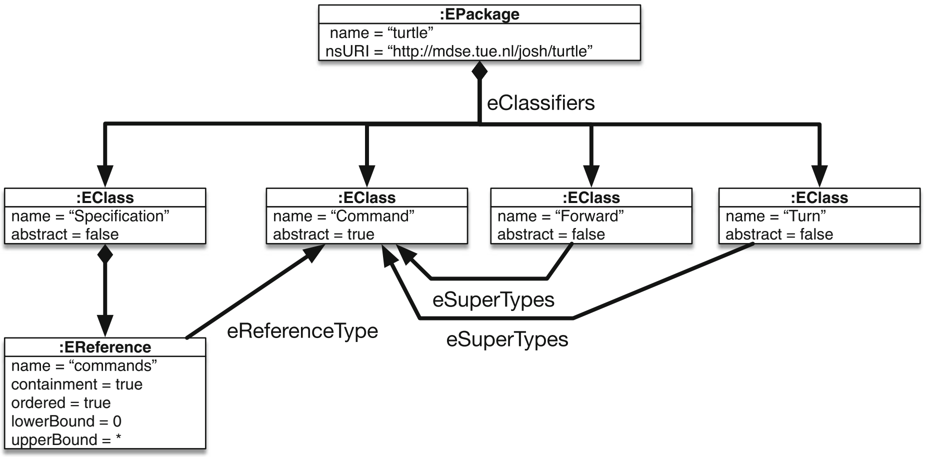
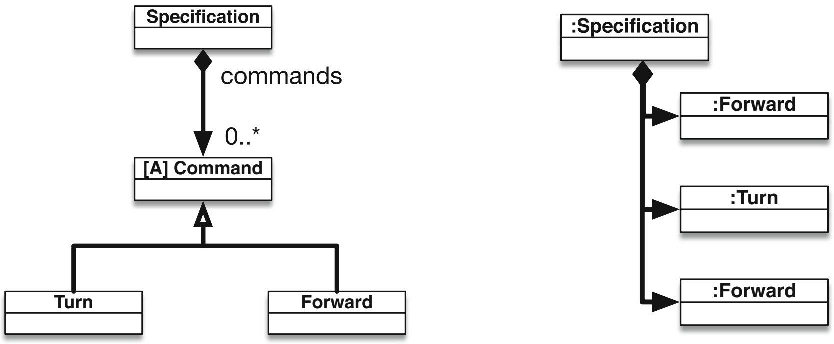


Fig. 3 The metamodel from Fig. 4a, in its abstract syntax. The notation used is similar to instance diagrams (also known as object diagrams). Note that such abstract syntax is seldom used, as its verbosity makes it illegible

2.4 The Object Constraint Language

Although Ecore provides a great deal of expressive power, there are times when an even more expressive formalism is needed. Consider again our model and metamodel example in Fig. 4. Say we want to prohibit our robot from making four consecutive turns. Ecore itself does not have ways to specify this. To mitigate this deficiency, dedicated formalisms have been created. The Object Constraint Language (OCL) is an OMG (1989) standard for specifying constraints on, amongst others, metamodels. One can use OCL to express, for example, requirement that a specification should contain an even number of commands



(a) A small DSL for describing movements of a robot

(b) Example of a model created using the metamodel from Figure 4a/Figure 3

Fig. 4 An example of a model and its corresponding metamodel

(Example 2.1), or prohibition to make four consecutive turns (Example 2.2). In general, an OCL constraint can be usually subdivided into three main parts (Warmer and Kleppe 2003):

1. The concepts on which the constraint operates, described by the **context** keyword. In Examples 2.1 and 2.2 below, the context is the `Specification` concept from the metamodel in Fig. 4a;
2. A property of the **context** on which the constraint is defined. In Example 2.1 this property is "commands" (cf. Fig. 4a). More commonly one writes "**self**.commands", as in Example 2.2;
3. The predicate to be evaluated on this property, e.g., "→size()→mod(2) == 0" in Example 2.1, describing that the size of that property is even.

■ **Example 2.1**

```
contex Specification:
commands→size()→mod(2) == 0
```

■ **Example 2.2**

```
contex Specification:
self.commands→not exists(c1, c2, c3, c4 |
    c1.oclIsTypeOf(Turn) &&
    c2.oclIsTypeOf(Turn) &&
    c3.oclIsTypeOf(Turn) &&
    c4.oclIsTypeOf(Turn) &&
    indexOf(c1) + 1 = indexOf(c2) &&
    indexOf(c2) + 1 = indexOf(c3) &&
    indexOf(c3) + 1 = indexOf(c4)
)
```

In OCL, it is also possible to define and call ad-hoc functions to assist in the specification of such constraints.

2.5 OCL in Transformations

The expressivity of OCL allows for a broader spectrum of applications than just specifying constraints. OCL can, for instance, also be used to specify model queries (Habela et al. 2008). Consider a hypothetical metamodel of a company and employees. We could use OCL to write a query to find all employees with the name Alice, as illustrated in Example 2.3. Such a query is little more than a constraint on the object we wish to find. By subsequently iterating every object in our "database" and evaluating the query against it, we can return every object for which that query evaluates to true.

■ **Example 2.3**

```
context Person
self.name = "Alice"
```

This application of OCL makes it well-suited both for model-to-model transformations (Section 2.5.1) and model-to-text transformations (Section 2.5.2).

2.5.1 Model-to-Model Transformations (m2m)

Model-to-model transformations (QVT 2015; QVTo 2015; Jouault and Kurtev 2006; Rose et al. 2010; Kolovos et al. 2008), are artifacts that transform models to other models, potentially instances of different metamodels. Model-to-model transformations can therefore be used to define the semantics of a DSL in terms of another DSL.

Let Fig. 5 present a metamodel `SpecificationV2` intended to replace the metamodel in Fig. 4a (`SpecificationV1`). Consider that a plethora of infrastructure is in place for defining (and potentially executing) models conforming to `SpecificationV1`. Redefining or reworking said infrastructure to conform to `SpecificationV2` would be time-consuming and costly. By transforming `SpecificationV2`-models into `SpecificationV1`-models, the infrastructure need not be adapted, whilst still adding all the benefits of such infrastructure for the `SpecificationV2` language. An example of such a transformation, called `Spec2toSpec1`, is given in Example 2.4.

The main method of `Spec2toSpec1` provides all root objects of models conforming to `SpecificationV1` (i.e., `SpecificationV1::Specifications`) are piped as inputs to mapping `spec2spec`. For every `SpecificationV2 Command` (`Up`, `Right`, `Down` and `Left`) this transformation creates a number of corresponding `SpecificationV1 Commands` (`Forward`, `Turn`). Such a mapping is in essence a transformation, not between models, but between objects. As part of implementation of `spec2spec` the helper method `desiredTurns` is used to determine the number of turns corresponding to `Left`, `Right`, `Up`, and `Down`. Another helper method, `numTurns`, given the current orientation of the robot determines the number of turns it has to perform to achieve the desired orientation. Helpers are similar to mappings, but do not have to return objects.

In the `spec2spec` mapping, every `Command` from the `SpecificationV2::Specification` is iterated using the OCL `iterate` construct. For each such `Command`, a number of `SpecificationV1::Turns` are appended to the "commands" reference of the implicitly defined `SpecificationV1`. Inside the `desiredTurns` helper, OCL is further used to for determining the type of the passed `Command`.

An example of an original model and transformed model obtained by applying `Spec2toSpec1` is illustrated in Fig. 6.

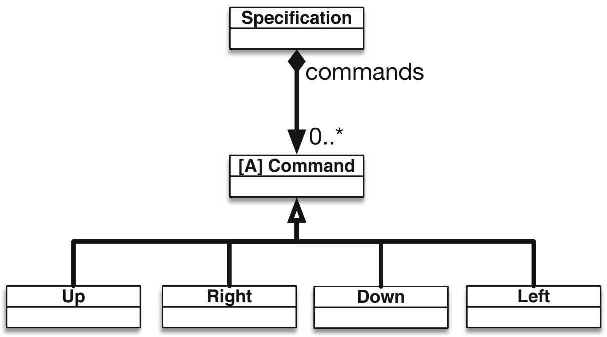


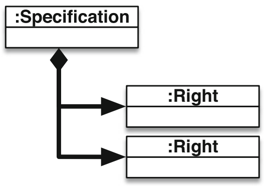
Fig. 5 A new version for the metamodel in Fig. 4a, which now works with absolute directions rather than relative ones

■ **Example 2.4**

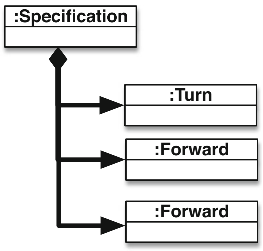
```
transformation Spec2toSpec1(in src : SpecificationV2, out dst : Specifi-
cationV1);
main() { src.rootObjects()[Specification] → map spec2spec(); }

    //Map a SpecificationV2 Specification to a SpecificationV1 Specifica-
tion
Mapping SpecificationV2::Specification:spec2spec() :
SpecificationV1::Specification {
    //Start direction is facing North.
    var direction = 0;
    self.commands → iterate ( c —
         forEach(1 .. numTurns(direction, desiredTurns(c))) {

             commands += Turn;
         }
         commands += Forward;
    );
}

    //Determine how many right-turns to take
helper numTurns(currentDirection : Integer, desiredDirection : Integer) : In-
teger {
     return (4 + desired - current) % 4;
}

helper desiredTurns(desiredDirection : SpecificationV2::Command) : In-
teger {
     if (c.oclIsTypeOf(Up)) {
         return 0;
     } else if (c.oclIsTypeOf(Right)) {
         return 1;
     } else if (c.oclIsTypeOf(Down)) {
         return = 2;
     } else if (c.oclIsTypeOf(Left)) {
         return 3;
     }
}
```

2.5.2 Model-to-Text Transformations (m2t)

A second type of transformation that OCL is used in is model-to-text (also referred to as code-generation) (MTL 2016). As suggested by the name model-to-text transform models to “flat” text. Although the generated text usually has some form of semantics and required structure, the transformation is not explicitly aware of these. As such, a model-to-text



(a) A model for SpecificationV2



(b) The result of transforming the Model in Figure 6a using the transformation in Example 2.4.

Fig. 6 Applying a model-to-model transformation to models

transformation can generate partial, or even invalid pieces of code if used incorrectly. As an example we could define a model-to-text transformation to transform a `Specification` into a small Logo (Muller 1998) program, without having knowledge of the underlying metamodel. Such a model-to-text transformation is illustrated in Example 2.5.

In Example 2.5 we use Acceleo-like syntax and wrap control structures in square brackets ([ ]). Execution of such a template is, similar to regular imperative programs, performed top-to-bottom. The `template` keyword associate the function `generate` with `SpecificationV2::Specification`. If such a `Specification` is passed as an argument to the template for generation, the instance is bound to the `specification` variable throughout the template. The `for`-loop then iterates over the elements in the `commands` reference of `specification`. For each such element the `if`-statements determine the type of the command and generate the corresponding Logo fragment. Whenever a non-control line (e.g., one without square brackets) is encountered, that line is added to the output file. The result of application of the model-to-text transformation in Example 2.5 on Fig. 6a yields the Logo program in Example 2.6.

■ **Example 2.5**
A model-to-text example using Acceleo-like Syntax

```
    [template public generate(specification : SpecificationV2::Specification)]
    [for (c : Command | specification.commands) ]
        [if c.oclIsType(Up)]
            setheading(0);
            forward(100);
        [/if]
        [if c.oclIsType(Right)]
            setheading(90);
            forward(100);
        [/if]
        [if c.oclIsType(Down)]
            setheading(180);
            forward(100);
        [/if]
        [if c.oclIsType(Left)]
            setheading(270);
            forward(100);
        [/if]
    [/for]
[/template]
```

■ **Example 2.6**
Result of a model-to-text transformation

```
setheading(90);
forward(100);
setheading(90);
forward(100);
```

3 Data Collection

In this section we start by describing the source of our data (Section 3.1) and its scope (Section 3.2) and then discuss the ways the data has been obtained and processed (Sections 3.3–3.4).

3.1 Data Source

To create a more representative, and up-to-date data set of OCL expressions 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 (Hebig et al. 2016; Kolovos et al. 2015) have also used the GitHub data. Finally, by focusing on GitHub we ensure that our data set includes the aforementioned OCL repository of Jordi Cabot that is also hosted on GitHub.

An alternative way of collecting data might have been using the Google search. Indeed, Google can provide access to meta-models stored on smaller sites, e.g., personal sites of researchers or their projects. However, we have observed that the indexing of GitHub files by Google is far from complete.

Lastly, Google BigQuery is a paid alternative for performing "analytics at scale" (e.g., large-corpus code searches). Unfortunately, the platform is paid. We believe that the process of constructing the dataset should be reproducible by every researcher interested in doing so. Therefore, when making choices pertaining to the design of our data collection approach we have explicitly excluded techniques that might incur costs.

3.2 Dataset Scope

The OCL expressions in our dataset are derived from metamodels and model-to-text transformations. OCL expressions belonging to metamodels can be stored either in a separate file or as part of a file containing the metamodel to which the OCL code refers. The naming convention of Eclipse, the most active open-source MDE community (Kolovos et al. 2015), requires extensions `.ocl` for OCL-only files and `.ecore` for meta-models. OCL expressions belonging to model transformations are always stored in the "model-to-text template" itself. These files have a `.mtl` file extension.

In this work we focus on files with `.ecore`, `.ocl`, and `.mtl` extensions. We exclude `.uml` from our scope as a large-scale dataset of UML models has already been made available by Robles et al. (2017). Furthermore, we exclude metamodels persisted with other extensions than `.ecore` (e.g., `.xmi`), as this is not the de facto standard for persistence.

3.3 GitHub Search

GitHub provides advanced search features, allowing one to look for different artifacts (e.g., commits, code files or wiki entries) that (1) match a given search string, (2) created during a given time period, and (3) owned by given users. Intuitively, to find OCL expressions from the meta-models 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. e.g., one cannot identify every file with a particular extension.

Kolovos et al. (2015) 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. (2015).

Hebig et al. (2016) used GHTorrent (Gousios and Spinellis 2012), to select 10% of GitHub repositories that are not forks and have a branch that can be downloaded (totaling 1240000 repositories). They then use a sequence of GitHub API calls to extract file lists for all these repositories to subsequently filter out relevant files. While this approach is very thorough, the sheer amount of API calls required, combined with the rate limit of GitHub makes the process very lengthy. In the case of Hebig et al. (2016), two weeks were required to gather all required data on only 10% of all GitHub repositories.

Therefore, we suggest an alternative approach to mining. To reduce the time required for data gathering, we seek to improve upon the method of Kolovos et al. (2015), but construct a query that does not suffer their limitations. Our solutions is to construct the search string by taking *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. (e.g., the query *extension:ocl foofoo* returned no results). Hence, the query *extension:ocl NOT foofoo* yields all files with the `.ocl` extension. By a similar argument *extension:mtl NOT foofoo* yields all files with the `.mtl` 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".[3] This value is persisted verbatim, and we use it to search for `.ecore` files containing OCL.

Thus, to identify OCL expressions we search on GithHub for the "code" using the following queries: (1) *extension:ocl NOT foofoo*; (2) *extension:ecore* "http://www.eclipse.org/emf/2002/Ecore/OCL"; (3) *extension:mtl NOT foofoo*.

In March 2017 the queries produced 6237, 1045, and 348677 hits, respectively. The extreme number of `.mtl` files is due to the `.mtl` extension also being used in 3D modeling (Material files) rather than model transformations. To reduce the sheer amount of false positives, we adopt the methodology of (Kolovos et al. 2015), and add the keyword "module" (marking the start of a model-to-text segment) when searching for `.mtl` files. This addition reduces the number of `.mtl` hits to 15379.

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 all related `.ecore` files. On GitHub, every *code match* belongs to a file in a repository. Rather than identifying single files, we

[3]See *Complete OCL tutorial* in Eclipse Neon documentation: https://help.eclipse.org/neon/index.jsp?topic=/org.eclipse.ocl.doc/help/CompleteOCLTutorial.html

download the entire repository containing files identified by our queries, and identify the files required for parsing OCL off-line.

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 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 the 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.

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 three lists of repositories, one for each of the search queries. We merge the lists and eliminate duplicates. This process yields a list of 671 relevant repositories.

3.4 Downloading and Stripping the Repositories

Using a Python script, we download the 671 relevant repositories. This step was performed on the July 31, 2017. As such, only revisions from before that date are included in the dataset. Next, we remove all files other than files ending in `.ocl`, `.ecore`, and `.mtl` as we only want to keep those for our data set, as well as directories that became empty as the result of this process. This results in a collection of of 6258 `.ocl` files, 21188 `.ecore` files, and 6783 `.mtl` files. In order to keep the files parsable we preserve the original file names and the directory structure.

3.5 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. For this uniqueness, we consider only the contents of the file, and exclude metadata and file names. The python script for performing this step is included in the dataset.

In the Ecore-based collection of 27446 (6258 + 21188) files, only 10894 files are unique.

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 (Willink 2011). We successfully parse 8947 files (82%) 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 in 504 AST files containing 9173 OCL expressions, derived from the Ecore-based section of our dataset.

To the extent of our knowledge, no abstract-syntax persistence for OCL in MTL exists. As such, the MTL-based part of our dataset only stores the raw data. Similarly to the

Ecore-based dataset, we perform duplicate removal, resulting in 6783 unique files, only 2634 of which were parsable. After eyeballing, this high rate of errors is due to the fact that many `.mtl` files found are still files related to 3D modeling, rather than model-to-text transformations.

4 Data Description

The dataset, in addition to the scripts written to compose it, is publicly available on GitHub.[4] To ensure persistence and permanent identification of the data we have registered it with the 4TU federation through DataCite Netherlands (Noten et al. 2017b).

4.1 Dataset Structure

The dataset consists of two collections:

- Ecore-metamodels and related `.ocl` files ("dataset");
- model-to-text transformations that contain OCL ("dataset-mtl").

Both collections contain a `repos` directory containing the raw data from the 671 repositories (245 `.ecore` and 349 `.mtl` repositories, with duplicates removed) identified in the previous section.

Immediate subdirectories of the `repos` directories are named after the repository's user or organization, and contain one or more directories corresponding to the repositories. These directories imitate the structure of the repositories. However, as explained in Section 3.4, all files other than `.ocl`, `.ecore`, and `.mtl` are removed, as well as directories that are empty as a result. For each repository in both collections, a `.json` file with metadata is included.

In addition to the `repos` directory the Ecore collection includes the `oclas` directory storing 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 (Willink 2011). These `oclas` files are named by the MD5 hashes of their contents, for faster equality-checking. Thus, `oclas` files for duplicate files are only persisted once. The relation between an `oclas` file and its original files is recorded in the `meta.json` in the dataset root.

Recall that in Section 3.5 we have eliminated multiple occurrences of files with the identical contents. Hence, we store the ASTs in files with names corresponding to their MD5 hash. This naming allows for faster comparison during processing. Each file with an AST has a file with the same name but `.json` extension, containing metadata about the repository and AST files.

Finally, the `meta.json` file relates the various `oclas` files in the `oclas` directory to their original files in the `repos` directory. For clarity, we illustrate the file structure, and its relation to the various GitHub repositories in Fig. 7. It does so using two keys in the root JSON object: `oclas` and `repos`. The `oclas` object maps the names of the AST files (without extension) 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.

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

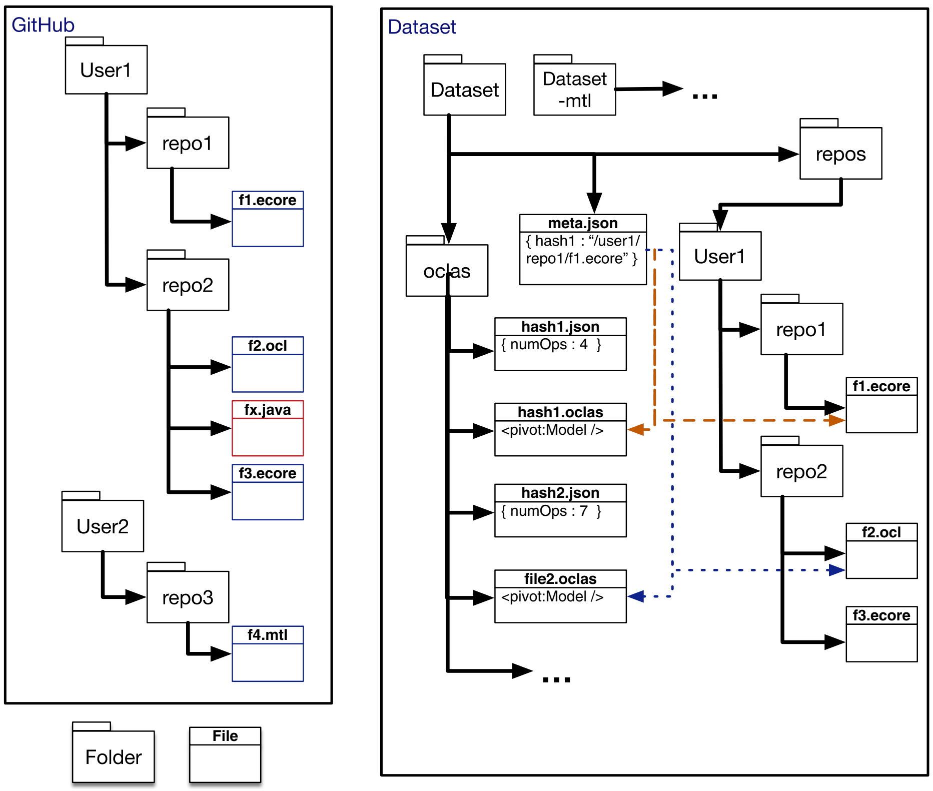


Fig. 7 A schematic overview of the file-structure of GitHub (left) and the representation of those files in our dataset (right)

4.2 Project Diversity

The OCL expressions we have collected are derived from repositories widely diverse in terms of size and activity. Table 1 summarizes sizes (KB) and the number of days between

Table 1 Size and activity of the repositories

	`.ecore` repositories (N = 245)		`.mtl` repositories (N = 349)	
	Size (KB)	Activity (days)	Size (KB)	Activity (days)
Min	0	0.0	0	0.0
Q1	567	0.0	242	0.0
Median	4831	16.0	1610	59.0
Mean	53522	218.5	47504	316.0
Q3	46257	277.0	23686	498.5
Max	981093	1977.0	1802744	2210.0

the repository creation date and the date of the most recent update (preceding the data collection).

4.3 Domains

To gain some preliminary insights into the domains, we perform frequent-word counts on the full filenames (including user and repository) for our OCL & Ecore dataset. We have manually analyzed the 400 most frequent terms appearing in the fully qualified filenames in the dataset. We decided to stop after 400 terms, as term ranked form 401 onwards occur only once. The top 30 results of this analysis are presented in Table 2.

Not surprising, terms such as org, eclipse, model, and ocl occur frequently. Important is to note that files that incorporate "examples" in their path occur extremely frequently.

Table 2 Top 30 most frequent words among the OCL & Ecore dataset

#	Word	num. occurences
1	eclipse	4325
2	model	3204
3	org	2896
4	ocl	2097
5	papyrus	1706
6	examples	1613
7	uml	1487
8	src	1472
9	robotml	1228
10	tests	1106
11	juancadavid	1079
12	impl	888
13	mdeforge	824
14	upr	790
15	extraplugins	753
16	xtext	719
17	ac	702
18	royalandloyal	600
19	refontouml	570
20	ontouml	521
21	bpmnprof	521
22	ecore	505
23	models	495
24	test	471
25	is	462
26	jp	462
27	pizzafactory	462
28	nces	462
29	a_rte	462
30	nagoya_u	462

To support these observations, we have performed card-sorting (Menzies et al. 2016) on every unique expression in our dataset to more precisely uncover the domains. From this we found the following domains: general-purpose modeling (UML, Petri nets, state machines), databases, web & networking, tutorial examples, student exercises, unit tests, academic paper examples, embedded, workflow modeling, and formal analysis.

We observed that examples and student assignments constitute a large part of our dataset. However, such repositories are not necessarily representative of real-world projects. As this may be a threat to validity to various studies based on our dataset, we wish to be able to separate the example files from the “real-world” ones. Following (Munaiah et al. 2017) we call the latter files “engineered” files, as they are more likely to represent sound software engineering practices. From the result of card sorting, we investigate the file paths of the examples and student exercise domains to extract terms that can be used to filter them. We compile the found terms into a single regular expression that can be used for matching:

“`/(example | assignment | praktikum | Übung | Uebung | Ejemplo | Profundizacion_Arquitectura_Software | practica | GenieLogiciel | Laboratorio | lecture | course)/i`”

A second regular expression is used to separate student exercises such as “ex4_week12”: “`/((NAMES[0-9\s_-]+)|([0-9\s_-]+NAMES))/i`” where `NAMES` is “`(td|tp|ws|ss|lab|GLS)`”.

Using this method, we can separate our 9173 unique expression ASTs from `.ecore` files into 6334 example ASTs and 8411 engineered ASTs. Note that the sum of 6334 and 8411 is higher than 9173 because unique ASTs can belong to multiple files.

We opt for this technique rather than a more elaborate approach of Munaiah et al. (2017). Indeed, the technique of Munaiah et al. includes or excludes entire repositories, while we reason at the level of individual files. Repositories that would have been classified as engineered in the style of Munaiah et al., but include a large number of example files (e.g., unit test models) would “contaminate” our data.

5 Studies Supported by the Dataset

As mentioned in the introduction, there are several possible applications of the dataset ranging from replication of earlier studies to comparison of OCL expressions derived from open source projects with those from closed-source projects. In this section we illustrate a number of these applications.

5.1 Replication Studies of Cadavid et al.

To show how our data set can be used in practice, we start by (partially) replicating the study of Cadavid et al. (2015).

5.1.1 The Original Study

To confirm their intuition that only a subset of the OCL is commonly used, and other parts of the language are rarely used, Cadavid et al. (2015) investigate what constructs from the OCL language are used in practice. To do so, they run various experiments on a dataset that they have compiled. The dataset consists of 24 metamodels derived from OMG standards, 11 metamodels from academic research, and 3 metamodels from industry. The metamodels come from a plethora of domains including databases, formal analysis and security.

In Section 2.3 of their work, Cadavid et al. (2015) present an overview of OCL concepts and relations between them. We reproduce this figure as Fig. 8 below. To show what parts of OCL are commonly and rarely used (Cadavid et al. 2015) measure the usage of every construct by counting the occurrences of every construct in their dataset of OCL expressions.

In the original study, Cadavid et al. (2015) study meta-modeling practices for OCL using on the OCL expressions from 37 meta-models. OCL data is also gathered from `.ocl` files, but no exact number is reported. In total of 995 OCL invariants (e.g., expressions) were found, 567 of which could be parsed.

5.1.2 The Original Findings

In the original study, Cadavid et al. (2015) create a histogram representing frequency of OCL constructs, reproduced below as Fig. 9a. They subsequently argue that the `OperationCall` construct is very generic, and is constituted by various sub-constructs (e.g., the various operations that are called). They create a similar histogram for operations which is reproduced as Fig. 10a. Furthermore, Cadavid et al. (2015) note that several operations in the the top 20 are defined in an ad-hoc manner (e.g., `getOutputFlows`, `getInputFlows`, `getFlows`).

5.1.3 The Replication Methodology

In order to verify whether the results of the original study generalize to a larger corpus, we perform an exact replication (cf. Shull et al. 2008) of the work by Cadavid et al. (2015). That

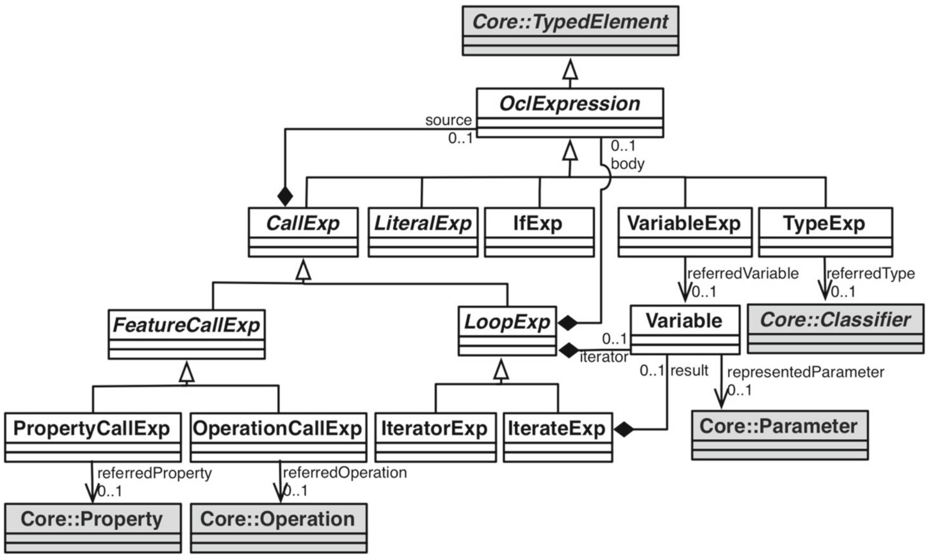
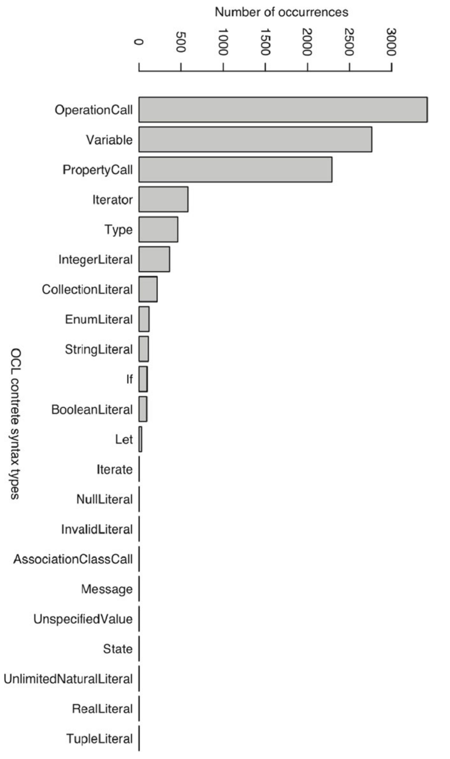
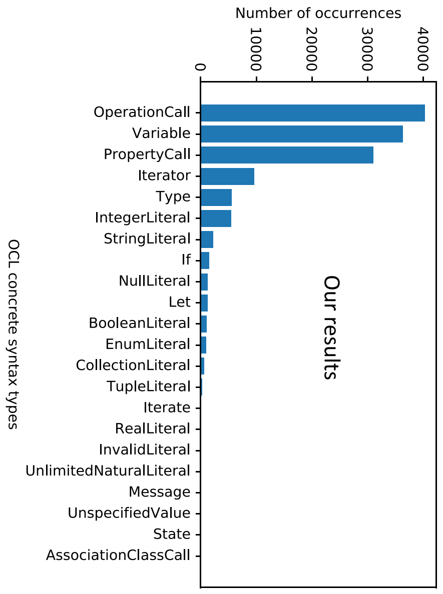


Fig. 8 A metamodel of the OCL language (Cadavid et al. 2015), showing the OCL concepts and relations between them. Classes with an italic name are abstract, where classes colored gray belong to other metamodels



(a) Observed distribution of used OCL constructs in the study of Cadavid et al (2015)



(b) Observed distribution of used OCL constructs in our full dataset

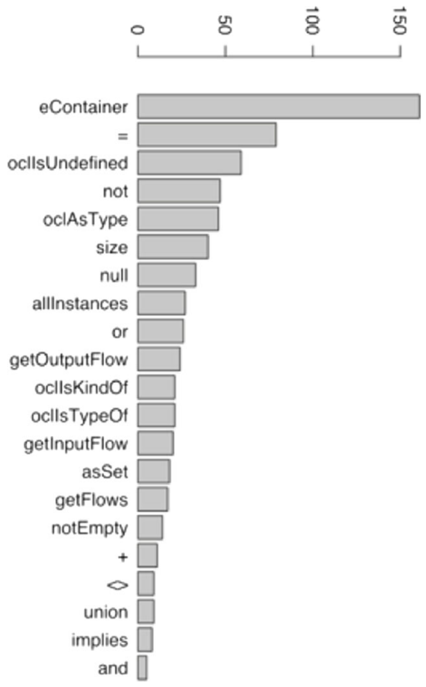
Fig. 9 Observed distribution of used OCL constructs in various datasets

is, we reuse the original methodology and apply it to our data set of OCL expressions. This dataset consists of 9173 unique expressions originating from 504 meta-models enriched with invariants from external `.ocl` files.

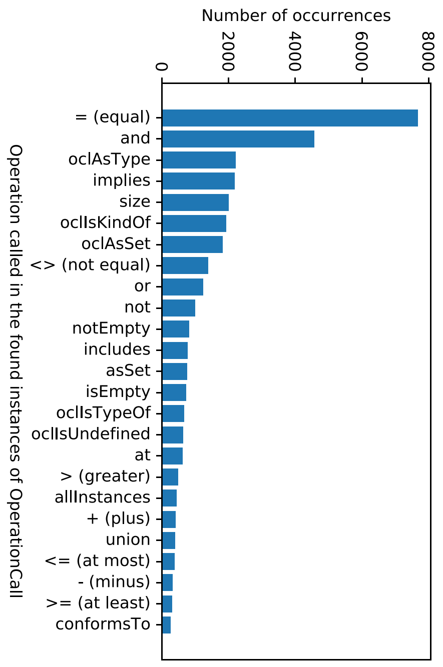
Similarly to the dataset of Cadavid et al. (2015), our dataset includes metamodels from a variety of domains, including databases, security, graphics, formal analysis, and workflow modeling.

In contrast to the dataset of Cadavid et al. (2015), our dataset consists only of Ecore-based metamodels and OCL expressions, whereas the dataset of Cadavid et al. (2015) also includes metamodels based on UML (1997), CMOF (Scheidgen 2006), and USE (Gogolla et al. 2007). Another major difference is that our dataset includes “example” files such as student projects. As such, we perform our experiments on the dataset as a whole, and on the engineered partition of our dataset (e.g., with examples filtered out).

Furthermore, our dataset is significantly larger than the original dataset of (Cadavid et al. 2015). One benefit gained through this increase in size, is that ad-hoc defined functions should no longer contaminate the most frequently used OCL constructs. (e.g., due to sheer



(a) Observed distribution of `OperationCalls` in the study of Cadavid et al (2015)



(b) Observed distribution of `OperationCalls` in our full dataset.

Fig. 10 Observed distribution of `OperationCalls` in various datasets

size, `OperationCalls` of ad-hoc defined functions will be subdued by generic OCL `OperationCalls`).

Absence of original numbers from the study of Cadavid et al. (2015) makes comparison difficult. As in the original study Cadavid et al. were predominately interested in the set frequently used OCL, we will inspect the overlap in the reported concepts. These comparisons will be limited to the number of concepts reported by Cadavid et al. (2015), as no other numbers are available. However, for our histograms, we will compute the top 25 most frequent `OperationCalls`. This should eliminate bias due to small differences in the distribution tails (e.g., a concept at rank 25 in the distribution of Cadavid et al. (2015) being at our rank 26).

Furthermore, as Cadavid et al. (2015) reports the numbers in a histogram, ranking (e.g., the position of each `OperationCall` in the top 20) of these concepts is also deemed important. To perform these comparisons, we will use Kendall's τ_b. Note that, the resulting p-values must be adjusted to compensate for performing multiple comparisons. We will do so using the method of Benjamini and Yekutieli (2001).[5]

[5]We only report already adjusted p-values

5.1.4 The Replication Findings

Using our EMF (Meta)Model Analysis tool (EMMA) (Mengerink et al. 2017b), we count the frequency of each OCL construct in all AST files. Figure 9b shows these frequencies. Computing Kendall's τ_b between this ranking and the original by Cadavid et al. (2015) (Fig. 9a) yields a result of $\tau = 0.685$ with a p-value too small to be computed exactly.

Like in the study of Cadavid et al. (2015), the `OperationCall` is the most frequently used construct. As such, following Cadavid et al., we take a closer look at `OperationCall` and create a histogram for frequencies of `OperationCall` constructs in the same way as Fig. 10a has been constructed in the original study. Figure 10b shows the resulting frequencies for our entire dataset.

Comparing the `OperationCalls` present, we find that they share 15/20 `OperationCalls`. Computing Kendall's τ_b between the `OperationCalls` of Cadavid et al. (2015) and `OperationCalls` in our full dataset yields $\tau = 0.0769$ with a p-value of $p = 1$.

As discussed in the methodology (Section 5.1.3), we conjecture that presence of a large amount of examples may be influence our results. As such we perform a similar comparison for both the "engineered" (cf. (Munaiah et al. 2017)) partition (Fig. 11a) and example partition (Fig. 11b) of our dataset. We find that both partitions share 16/20 concepts with the results of Cadavid et al. (2015) (Fig. 10a).

Computing Kendall's τ_b between the `OperationCalls` of Cadavid and those of our engineered and example partitions yields $\tau = -0.0513$ and $\tau = 0.103$ respectively, both with an (adjusted) p-value of $p = 1.0$.

These results make us believe that the engineered and example partitions exhibit little differences. Computing Kendall's τ_b between both partitions yields $\tau = 0.744$ with an (adjusted) p-value of $p = 0.004217$. Furthermore, 21/25 concepts are shared between them.

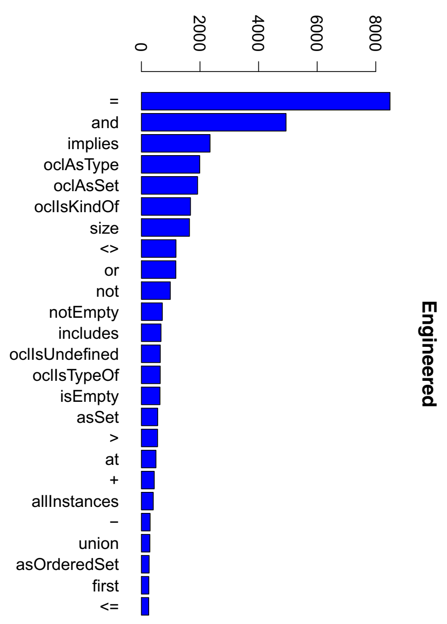
5.1.5 Discussion

Comparing the most used constructs by Cadavid et al. (2015) (Fig. 9a) to the most used constructs for our full dataset (Fig. 9b), we find that the results are highly similar. The sets of reported constructs are even identical (e.g., the 21 constructs reported by Cadavid as most frequent are also our 21 most frequent constructs). These results are in line with the original results by Cadavid et al. (2015): e.g., the six most used constructs are identical even with respect to their order (`OperationCall`, `Variable`, `PropertyCall`, `Iterator`, `Type` and `IntegerLiteral`).

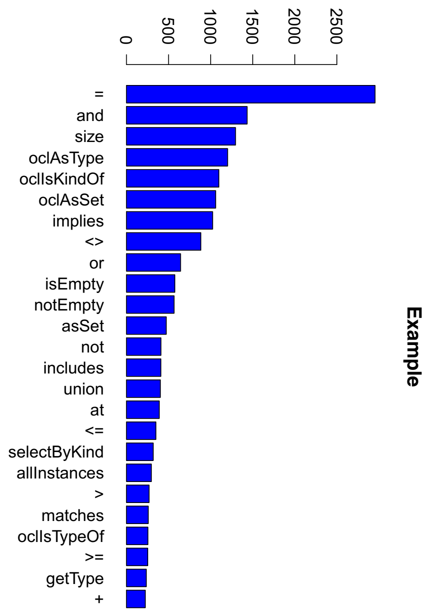
Like in the study of Cadavid et al. (2015), the `OperationCall` is the most frequently used construct in our dataset (Fig. 9b). Concurring with Cadavid et al. (2015), we expected this outcome, as `OperationCall` encapsulates all operations that are called on OCL constructs (e.g., AND, OR, forEach).

Further inspection of the distribution of our `OperationCalls` (Fig. 10b), shows that our results are *not* in line with the results of Cadavid et al. (2015) (Fig. 10a). Although 75% (15/20) `OperationCalls` observed are shared between distributions, their rankings show no similarity ($tau = 0.04769$). Both the original ($p = 0.76$) and adjusted ($p = 1.0$) p-values indicate that the distributions are independent.

Recall form Section 5.1.1 that the dataset used by Cadavid et al. makes use of data from standards, scientific, and industrial sources. Moreover, the domains from which their data is drawn are similar to those in the engineered partoitition of our dataset (e.g., databases, workflow modeling, security). The most significant difference between the dataset of Cadavid



(a) Observed distribution of `OperationCalls` in the “engineered” partition of our dataset.



(b) Observed distribution of `OperationCalls` in the “example” partition of our dataset.

Fig. 11 Observed distribution of `OperationCalls` in various datasets

et al. (2015) and ours, is that our dataset includes a large amount of examples and student projects. As we have expected (Section 5.1.3), the large amount of example files in our dataset may lead to a difference between rankings, specifically between Fig. 10a and b

As such, we replicate the `OperationCall` histogram for the “example” and “engineered” (cf. (Munaiah et al. 2017)) partitions of our dataset separately. The results are shown in Fig. 11a and b, respectively.

As such, we compare the distribution of `OperationCalls` from the dataset of Cadavid et al. to the distributions of `OperationCalls` from the OCL expressions in the engineered partition (Fig. 11a). Although the domains of Cadavid et al. (2015) and our “engineered” partition are more similar than when compared to our “example” partition, both our distributions (Fig. 11a and b) share 16/20 `OperationCalls`. Moreover, the same 4 `OperationCalls` are dissimilar.

Computing Kendall’s τ_b between the `OperationCalls` of Cadavid et al. (2015) (Fig. 10a) and the engineered and example partitions respectively yields $\tau = -0.0513$ and $\tau = 0.103$, respectively, both with (adjusted) p-values of $p = 1.0$ (originally $p = 0.85478$ and $p = 0.66933$, respectively).

The dissimilarity between (Cadavid et al. 2015) and engineered partitions surprises us as its domains are more similar than that of the example partition. Indeed, both datasets have similar domains. Following the argument of Cadavid et al. (2015) that `eContainer` is not part of the OCL language itself, but part of one of its standard libraries (cf. `stdio.h` in C), we exclude it from further analysis.

By manually comparing the ranks of `OperationCalls` in both distributions (Table 3) we made the following observations.

The largest difference between both rankings is in the boolean operators `and` and `implies`. In particular the increase in rank of `and` and similar decrease in rank of `allInstances` (+18 and -13, respectively) is noteworthy. The `allInstances` `OperationCall` is typically found in the root of a constraint, e.g., `allInstances` marks a new context for an OCL expression.

■ **Example 5.1**

Person.allInstances()→forAll(self.age > 18)
Person.allInstances()→forAll(self.height > 180)

Table 3 The absolute and relative positions of `OperationCalls` from Figs. 10a and 11a respectively, ordered largest relative (absolute) difference first

`OperationCall`	Cadavid et al. (2015)	Engineered partition	Difference
and	20	2	+18
implies	19	3	+16
allInstances	7	20	−13
oclIsUndefined	2	13	−11
<>	17	8	+9
not	3	10	−7
union	18	22	−4
oclIsKindOf	10	6	+4
oclIsTypeOF	11	14	−3
asSet	13	16	−3
+	16	19	−3
size	5	7	−2
or	8	9	−1
oclAsType	4	4	+0
=	1	1	+0
getOutputFlow	9	NA	NA
getInputFlow	12	NA	NA
getFlows	14	NA	NA
notEmpty	15	NA	NA
null	6	NA	NA
includes	NA	12	NA
isEmpty	NA	15	NA
at	NA	18	NA
−	NA	21	NA
asOrderedSet	NA	23	NA
first	NA	24	NA
<=	NA	25	NA

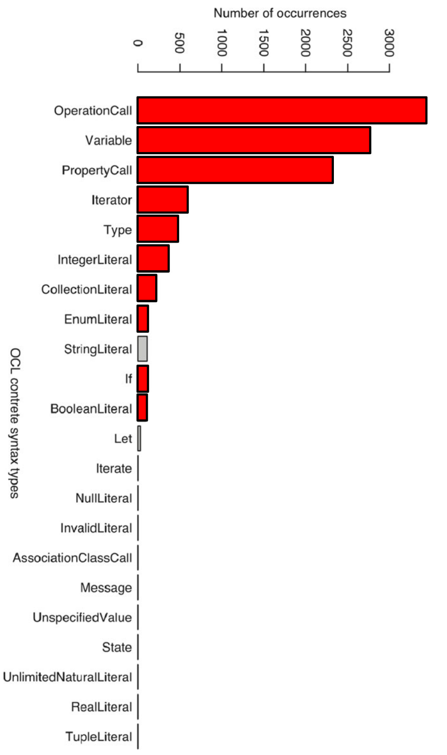
Decrease in the rank of `allInstances` suggests that in the engineered partition of our dataset introduction of a context occurs less frequently. Indeed, instead of introducing the same context several times and defining the corresponding OCL expressions per context, one could combine multiple OCL expressions sharing the same context using `and`.

■ Example 5.2
Person.allInstances()→forAll(self.age > 18 and self.height > 180)

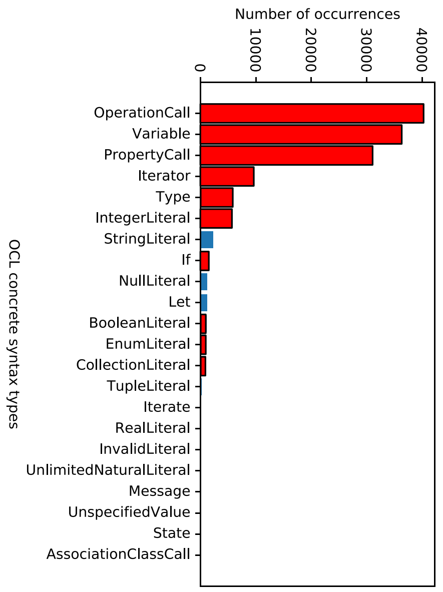
Preference for combining OCL expressions would, subsequently, results in the higher number of `and`s.

In Section 6.1.4 of the original work by Cadavid et al. (2015) the authors list ten constructs that capture 98.6% of all OCL expressions, as illustrated in Fig. 12a. Counting the number of OCL expressions captured by the same ten constructs in our dataset yields merely 68.3%, as illustrated in Fig. 12b.

Furthermore, note that `getOutputFlow`, `getInputFlow`, and `getFlows`, present in Fig. 10a, are absent from Fig. 11a. Indeed, Cadavid et al. (2015) explain that



(a) 10 OCL Constructs named in the study of Cadavid *et al.* Cadavid et al (2015), that allow for the specification of 98.9% of OCL expressions in their dataset, projected onto Figure 9a in red;



(b) The same 10 concepts from the Study of Cadavid *et al.* (Cadavid et al (2015), Figure 12a), projected onto Figure 9b in red. Here only able to specify 68.3% of expressions.

Fig. 12 Differences in the findings of Cadavid, and our findings

these are methods that are defined ad-hoc within the OCL expressions. Such ad-hoc defined `OperationCalls`, also present in our dataset, are subdued by generic `OperationCalls`: the first ad-hoc `OperationCall` in the engineered partition of our dataset occurs at rank 63.

The remaining `OperationCalls` from the top 20 of Cadavid et al., notably absent from our top 25 are `null` (that might have been caused by resolution errors during the parsing step in the original study) and `notEmpty` (whose place is taken by `isEmpty` in the engineered partition of our dataset).

Lastly, equivalence checking (=) is still the most used construct. This seems logical, as equivalence checking is what constraints are all about.

Summarizing, the set of OCL most frequent `OperationCalls` identified by Cadavid et al. (2015) is, to a large extent, the same in the our dataset. Although their positions have shifted, 15 concepts from the top 20 of Cadavid et al. (2015) are still in the top 25 of our full dataset.

Going beyond the context of MDE, we observe that all distributions observed (Figs. 9a and b, 10a and b, 11a and b) exhibit a Pareto-like shape that is common in studies related to refactoring (Murphy-Hill et al. 2012) and, broader, software engineering (Goeminne and Mens 2011).

5.2 Benchmarking

A major issue with empirical research into model-driven engineering is that the amount of real-life data is scarce (Gogolla et al. 2013; Gogolla and Cabot 2016). Most research revolves around standards documents or small corpora of open-source data.

Industrial data is even more scarce, as most companies consider their MDE data proprietary. This has several downsides:

– Checking results and findings of studies performed on these closed-source datasets is impossible;
– Generalization of techniques and results from industrial data are unclear.

We propose to use our large-scale open-source OCL dataset as a benchmark. First, studies originally performed on closed-source (industrial) datasets can be *replicated* on our dataset evaluating generalisability of the results. Alternatively, one can *compare* properties of closed-source OCL expressions with properties of OCL expressions in our dataset. By performing such a comparison one might either conclude that the closed-source OCL expressions are similar to the ones in our dataset and therefore more advanced conclusions derived from analyzing our dataset can be transferred to the closed-source OCL expressions; or the closed-source OCL expressions are inherently different from the ones in our dataset requiring a more profound investigation of the differences between the two.

5.2.1 Open-Source Versus Closed-Source

As proof-of-concept we have compared a closed-source dataset of OCL expressions and the OCL expressions in our dataset with respect to their complexity. However, as observed by Munaiah et al. (2017), some open source projects on GitHub are merely examples or student projects rather than engineered software projects. Therefore, in this section we compare complexity of *three* groups of OCL expressions: those on closed-source metamodels, those on example metamodels from GitHub and those on engineered metamodels.

Discussion in this section extends our earlier work (Mengerink et al. 2017a) by distinguishing example and engineered GitHub metamodels, and consequently performing more advanced statistical analysis.

Methodology We compare the engineered partition of our dataset to a closed-source (proprietary) dataset obtained from ALTRAN (1982), a large company offering third-party MDE services. The 93 expressions in the dataset are derived from seven metamodels.

Having identified which files are "engineered" or "examples", we use EMMA IWSM-Mensura2017 to compute the complexity values for all expressions. To compare complexity of the engineered partition, example partition, and closed-source OCL expressions we employ the complexity metric introduced by Cadavid et al. (2015) for the complexity of OCL Expressions (Definition 7 and Listing 12 Cadavid). This metric is defined as the the number of metamodel elements that the expression uses, an approach that is also found elsewhere cabotcomplexity. Although other metrics may exist, we adhere to the metric of Cadavid et al. (2015), as it is used in the literature and is easy to understand.

After the complexity measurements have been performed we need to compare three distributions of complexity values. Traditionally, a comparison of multiple groups follows a two-step approach: first, a global null hypothesis is tested, then multiple comparisons are used to test sub-hypotheses pertaining to each pair of groups. The first step is commonly carried out by means of ANOVA or its non-parametric counterpart, the Kruskal-Wallis one-way analysis of variance by ranks KruskalWallis. The second step uses the *t*-test or the rank-based Wilcoxon-Mann-Whitney test mannwhitneywilcoxon, with correction for multiple comparisons, e.g., Bonferroni correction Dunn1961,paramnonparam or the method of Benjamini and Yekutieli (2001). Unfortunately, the global test null hypothesis may be rejected while none of the sub-hypotheses are rejected, or vice versa Gabriel. Moreover, simulation studies suggest that the Wilcoxon-Mann-Whitney test is not robust to unequal population variances, especially in the case of unequal sample sizes BIMJ:BIMJ17,Zimmerman:Zumbo. Recall, that the collection of OCL over engineered open-source metamodels is more than 100 times larger than the collection of OCLs over closed-source metamodels; similarly, variance of complexity for OCL over engineered open-source metamodels is more than five times larger than variance of complexity of OCLs over closed-source metamodels.

Therefore, one-step approaches are preferred: these should produce confidence intervals which always lead to the same test decisions as the multiple comparisons. We use the $\widetilde{\mathbf{T}}$-procedure of Konietschke et al. (2012). This statistical procedure can perform different kinds of intergroup comparisons: we focus on the comparisons between all pairs of distributions; such a comparison is known as a Tukey-type contrast Tukey. Furthermore, similarly to the previous studies we use the default probit transformation and the traditional 5% family error rate (cf. VasilescuESE13,YuWYW16,SwidanSH17,CasseePCS18). A more detailed discussion of the application of the $\widetilde{\mathbf{T}}$-procedure to software engineering data and an illustrative example can be found in the article of Vasilescu et al. (2013).

To complement the study for statistically significant differences we also report the effect size using Cliff's delta Cliff.

Results Distributions of the complexity values of closed-source and open-source OCL expressions are shown in the violin plots violinplot in Fig. 13. The $\widetilde{\mathbf{T}}$-procedure shows that

- OCLs over engineered metamodels tend to be more complex than over the example metamodels (p is too small to be computed precisely);

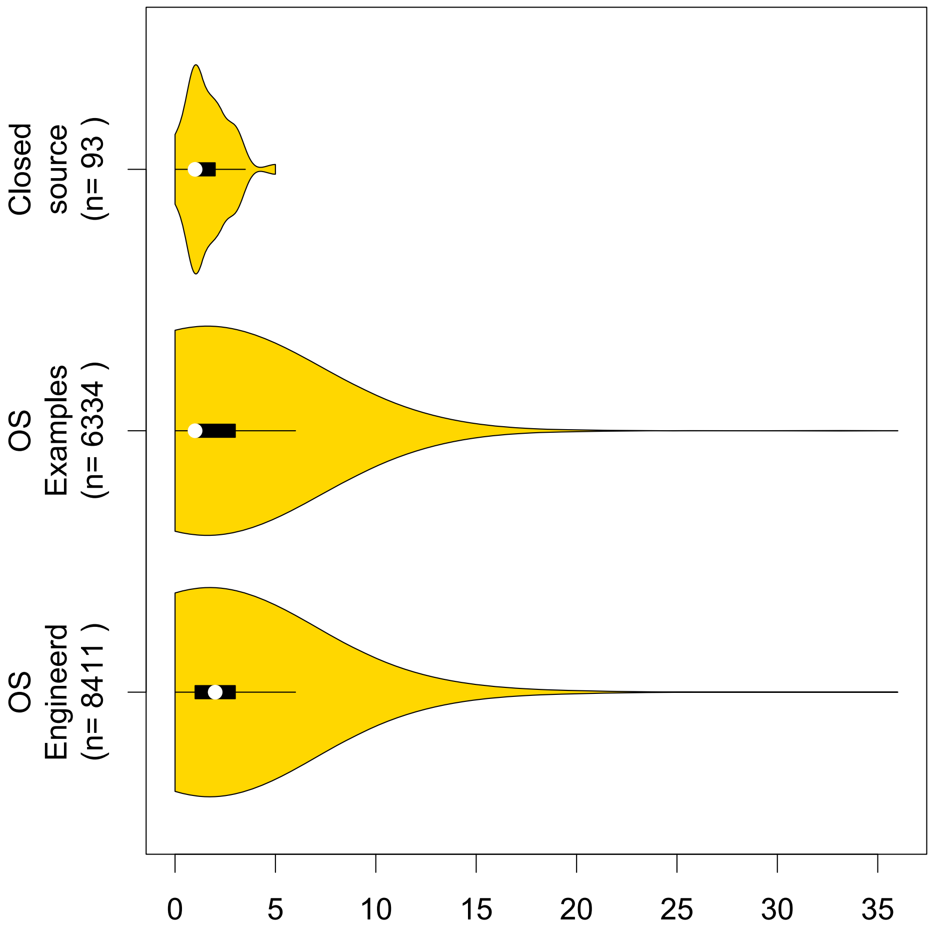


Fig. 13 Violin plots of the complexities of closed-source Ecore-based OCL, and open-source Ecore-based OCL, reproduced from Mengerink et al. (2017a). Note that the numbers are higher than reported in Section 4, because unique expressions may occur in both example and engineered projects

- OCL over the closed-source metamodels tend to be more complex than over the example metamodels ($p \sim 0.02$);
- no statistically significant relation could be established between the OCLs over example metamodels and the closed-source metamodels ($p \sim 0.99$);

While the differences between the complexities of OCLs over the engineered metamodels and the example metamodels, as well as between those over the engineered metamodels and closed-source metamodels are statistically significant the effect size is negligible (cf. Romano et al. 2006): ca. 0.12 and 0.13, respectively.

Discussion We observe that while two of the three pairs are statistically different the effect size is negligible. i.e., differences in complexity of OCL expressions over metamodels from open-source engineered projects, open-source example projects and closed-source projects are practically unimportant. We therefore conclude that OCL expressions over open-source engineered metamodels on GitHub, collected in our dataset, can be used as a proxy for studying complexity of OCL expressions both over open source example metamodels an, more importantly, over metamodels from closed-source projects. Moreover, we conjecture that the same conclusion would hold for other metrics defined for OCL expressions. Validity of this conjecture should be subject of follow-up studies.

Threats to Validity As any empirical study, discussion in this subsection is subject to several threats to validity. We postpone their discussion to Section 7.

Future Work Future work would be to involve closed-source meta-models coming from a larger number of companies, to strengthen the statistics for this experiment. However, as industry is quite protective of its models, we envision obtaining a broad and representative set to be tedious if not impossible.

5.2.2 *Ecore Versus Acceleo*

Following the results from Section 5.2.1, we assess to what extent results obtained in EMF-based OCL context could be generalized to other contexts. In this study we use both the Ecore-based and MTL-based (Acceleo) collections of our dataset.

Methodology As above we perform the Mann-Whitney test using $p = 0.05$ as a threshold, with the following hypotheses:

- H_0: The distributions of complexity of the samples of Ecore-based and MTL-based OCL expressions represent two populations with the same median values.
- H_{alt}: The distributions of complexity of the samples of Ecore-based and MTL-based OCL expressions represent two populations with different median values.

As in Section 5.2.1, we complement the study by using Cliff's delta for effect size.

Results Distribution of complexities of OCL expressions derived from meta-models and model transformations is illustrated in Fig. 14. The p-value associated with the Mann-Whitney test is too small to be computed precisely, $p < 2.2 \times 10^{-16}$, e.g., H_0 can be rejected. Computing the effect size using Cliff's delta yields a value of 0.568 with a 95% confidence interval of [0.557, 0.578]. This effect size can be interpreted as large (Romano et al. 2006).

Discussion With a p-value too small to compute precisely, there is a statistically significant difference between complexity in Acceleo and Ecore-based OCL. Moreover, the effect size is large suggesting that OCL expressions from open-source Ecore are substantively more complex than those from open-source MTL. This is to be expected, as the roles OCL takes on in both contexts are different. In Ecore, OCL is primarily used to define constraints on abstract-syntax structures (e.g., `Context graph: self.outDegree > 1`). Whereas in Acceleo, OCL is used primarily to define queries and selectors for transformations (e.g., `ePackage.eClassifiers->filter(EClass))`).

The conclusion from this experiment is thus, that results obtained from Ecore-based OCL may not necessarily be generalized to Acceleo, and vice-versa. Based on this observation we revisit the discussion of the frequency of the OCL constructs (cf. Figs. 8 and 9b) and investigate the distribution of the frequency of the OCL constructs for four different

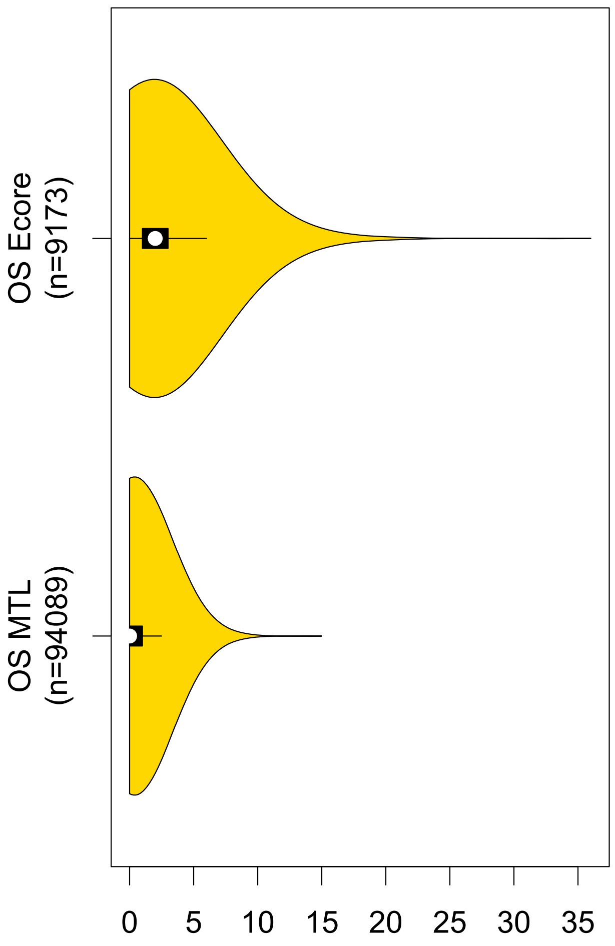


Fig. 14 Violin plots of the complexities of OCL in Ecore- versus Acceleo-based OCL

categories of OCL expressions: those derived from open-source meta-models, closed-source meta-models, open-source transformations and closed-source transformations. OCL expressions from open-source meta-models and open-source transformations constitute our dataset; those from the closed-source meta-models have been discussed in Section 5.2.1 and those from the closed-source transformations cannot be made public due to confidentiality reasons. Figure 15 summarizes the normalized usage frequencies, e.g., percentages of usage. We opt for percentages such that we could compare the usage for different categories.

Figure 15 shows that the usage of OCL constructs differs heavily per chosen category of OCL expressions, e.g., `Variable` is the most frequent construct in closed-source (industrial) transformations, while `StringLiteral` occupies the highest share among the OCL constructs. This suggests that results obtained for one category of OCL expressions are not necessarily transferable to a different category.

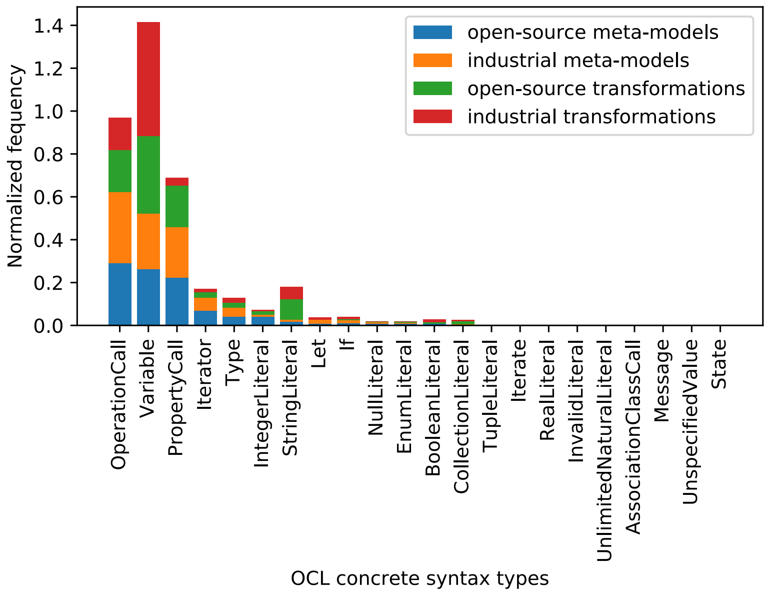


Fig. 15 Normalized usage of various OCL constructs: closed-source (industrial) meta-models, closed-source (industrial) model-to-text transformations (mtl), open-source meta-models, and open-source model-to-text transformations (mtl). The blue bars (open-source meta-models) correspond to Fig. 9b

5.3 Limiting Threats to Validity of Another Study

In this subsection we show how the dataset compiled can be used to evaluate assumptions made by previously published analysis techniques. Indeed, if those assumptions are frequently challenged by the real OCL code, more advanced analysis techniques should be developed; if, however, these assumptions hold in the lion's share of OCL code, then they can be safely made: a similar argument has been recently made by Landman et al. (2017) for static analysis of Java programs using reflection, and by Casalnuovo et al. (2015) for usage of the C/C++ `assert` construct. Both (Landman et al. 2017) and (Casalnuovo et al. 2015) used GitHub data to evaluate validity of the assumptions.

To illustrate this application of the dataset we consider the work of Anastasakis et al. (2008) and evaluate practical relevance of the assumptions made. Anastasakis et al. (2008) advocated analysis of UML models by first transforming them to Alloy (Jackson 2012), and then applying constraints solving techniques to the resulting models. However, the authors also observed that various OCL concepts cannot be expressed in Alloy, possibly rendering their technique inapplicable in practice.

The leading example of discrepancy between UML and Alloy is the `Iterate` operation, which has no equivalent in Alloy. The `Iterate` concept is used for imperative-style iteration over collections, however Alloy is a declarative-oriented language, and as such no generic translation exists.

By consulting our dataset with respect to the `Iteratre` construct, we observe that:

– 8% (19/245) of repositories use the `Iterate` construct;

- only 5% of (24/504) of all meta-models within these projects use the construct. It thus appears that the number of meta-models per project that use `Iterate` is limited;
- at an even larger scale, less than 1% (82/9173) of OCL expressions make use of the `Iterate` construct. So even within the limited number of meta-models that use the construct, the amount of OCL expressions that require it also limited.

Hence, most OCL expressions are not affected by the limitation identified by Anastasakis et al. (2008), and the technique of Anastasakis et al. (2008) can be applied to almost all expressions.

6 Related Work

The need for replication on larger dataset from a broad context of domains publicly has been listed as future work for many empirical study in MDE; more recently this need has been also recognized by researchers in MDE (Gogolla et al. 2013; Gogolla and Cabot 2016).

As explained in the Introduction most of the research so far has either considered prepared datasets rather than the real-world ones (e.g., the work of Reynoso et al. 2006 and Correa et al. 2007) or relatively small collections of OCL expressions (e.g., the work of Correa et al. 2015, Cabot,[6] and Basciani et al. 2014). The current submission is based on and extends two of our earlier publications (Noten et al. 2017a; Mengerink et al. 2017a). In Noten et al. (2017a) we have described the data collection process, presented the dataset of OCL expressions derived from the Ecore meta-models, replicated a study of Cadavid et al. (2015) and evaluated the assumptions made by a previous work of Anastasakis et al. (2008). In the follow-up work (Mengerink et al. 2017a) we have compared complexity of OCL expressions derived from open-source and industrial meta-models.

In particular, compared to these publications in the current work we investigate a more diverse set of files including model transformations.

7 Threats to Validity

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

Construct Validity Construct validity is the degree to which the test used measures the construct the test has been designed to measure. For instance, in our study of Open-Source versus Closed-Source OCL expressions in Section 5.2.1 we follow (Cadavid et al. 2015) and operationalize complexity of an OCL expression as the number of meta-model elements that the expression uses. While this approach is also found elsewhere (Cabot and Teniente 2006), it provides a very restricted view on the notion of complexity, and therefore, introduces imminent threats to construct validity. Indeed, structure of an OCL expression or names of the meta-model elements referenced can be expected to be related to the expression complexity in the same way as structure of a function and names of the variables used are related to complexity of traditional software systems. We consider, however, designing

[6]https://github.com/jcabot/ocl-repository

a validated complexity measure for OCL expressions to be a challenge of its own going beyond the scope of this article. artofzThe dataset we provide can be used to assess validity of such a complexity measure.

Furthermore, our operationalization of "example" metamodels assumes that the fully qualified package name contains an indication whether the project is an example or not, and is based on manual identification of terms-related to examples among terms frequently used in the fully qualified package names. To reduce the impact of this threat, the labeling was carried out by the first author, having a masters degree in computer science.

Internal Validity Internal validity is related to the inference of the conclusions based on the data collected.

Validity of our conclusions might have been threatened by our decision to *use GitHub*, e.g., due to the limitations of the *search functionality of GitHub* (git 2008).

The search functionality of GitHub only allows searching of the main branch in repositories, e.g., our search query might miss files (Bird et al. 2009). 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.

Yet 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. Plotting the number of files per repository in our dataset (Fig. 16) reveals that there are two major outliers at approximately 80,000 and 100,000 files. This indicates that the probability of missed repositories (e.g., repositories with over 500,000 files) is slim.

Specifically, internal validity of the example studies in Section 5 might also have been threatened by the choice of the statistical machinery. We have used well-established statistical techniques that have been successfully applied to software engineering.

External Validity External validity pertains to generalization of the conclusions derived beyond the context of our study. While we do not claim our findings to be valid beyond GitHub, 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 major concern for external validity of our study in Section 5.2 is related to the fact that our collection of OCL expressions over metamodels from closed-source projects is small, and that all these projects have been carried out by the same organization. Unfortunately, real-life MDE data is scarce (Gogolla et al. 2013; Gogolla and Cabot 2016) and real-life closed-source data is even less accessible for the researchers. Therefore, we would like to encourage replication of our study (Shull et al. 2008) with the goal of confirming or circumscribing its conclusions.

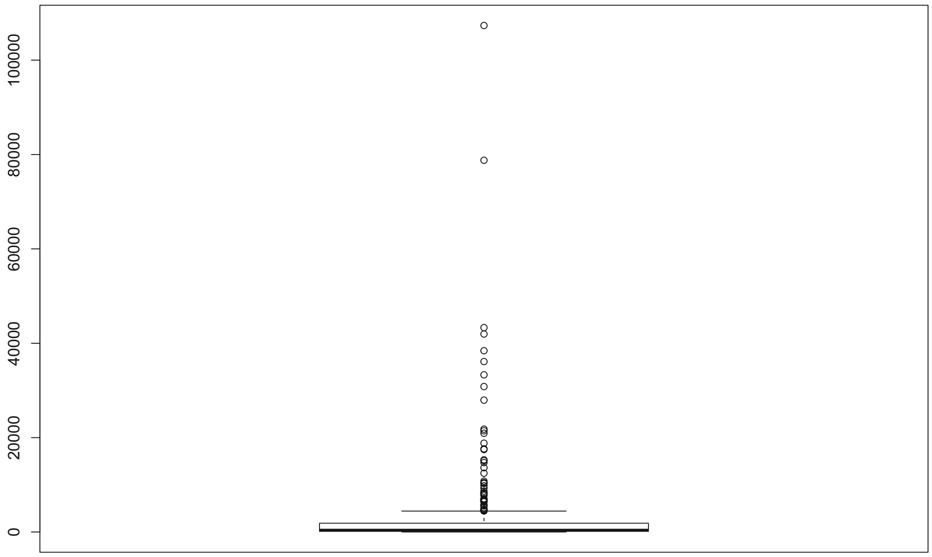


Fig. 16 A boxplot of the number of files per repository

8 Conclusions

In this work we present a publicly available dataset of OCL expressions derived from GitHub. The dataset is composed of two collections of OCL expressions:

– A collection of 9173 OCL expressions, derived from 504 unique `.ocl` and `.ecore` files, originating from 245 systematically selected GitHub repositories (Noten et al. 2017a)
– A collection of 94089 OCL expressions, derived from 2634 unique `.mtl` files originating from 349 systematically selected GitHub repositories.

The data set includes the original `.ocl`, `.ecore`, and `.mtl` files, as well as the generated AST files for the OCL/Ecore dataset. The AST files are stored in XMI format conforming to the OCL Pivot Meta Model. Furthermore, lists of used repositories, as well as a variety of metadata for the various files is provided.

This data set allows for a variety of empirical studies of the OCL, including usage studies and practical evaluations of proposed techniques. We have performed several case studies of various types to illustrate the applicability of this data set in practice. As such, we highly encourage the reader to download the dataset,[7] as we consider the primary value of this work to be the dataset itself, not the example experiments we have performed with it.

In particular, we have extended the previous version of our dataset (Noten et al. 2017a) with OCL expressions drawn from Acceleo-based model-to-text transformations. This adds a plethora of possible new studies, a hint of which we have presented in Section 5.2.

Throughout this work, we have already hinted at various pieces of future work with respect to the experiments performed. However, as stated, the primary value of this work is the dataset itself, and as such most of the future work will be in improving it.

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

Firstly, extending the dataset with an even broader set of models is a main piece of future work. One could consider different sources such as Google Code, SourceForge, but also data from bug-trackers or mailing lists could be considered.

Furthermore, OCL is used in a broader scope than just Ecore or model-to-text. One could also include models that use OCL to implement operations, or model-to-model transformations that also use OCL. The list of repositories that were used for mining MTL- and Ecore-based data should serve as a valuable research for enabling such extensions.

Acknowledgements The work of the first author has been funded by the ASML Holding N.V.

References

ALTRAN (1982) ALTRAN. https://www.altran.com. Accessed 10 April 2018

Anastasakis K, Bordbar B, Georg G, Ray I (2008) On challenges of model transformation from UML to Alloy. Software &, Systems Modeling 9(1):69–86

Basciani F, Di Rocco J, Di Ruscio D, Salle AD, Iovino L, Pierantonio A (2014) MDEForge: an extensible web-based modeling platform. In: CloudMDE@MoDELS, CEUR-WS, pp 66–75

Benjamini Y, Yekutieli D (2001) The control of the false discovery rate in multiple testing under dependency. The Annals of Statistics 29:1165–1188

Bézivin J (2006) Model driven engineering: An emerging technical space. In: Generative and Transformational Techniques in Software Engineering, Lecture Notes in Computer Science, vol 4143, Springer, pp 36–64

Bird C, Rigby PC, Barr ET, Hamilton DJ, Germán DM, Devanbu PT (2009) The promises and perils of mining git. In: Mining Software Repositories, pp 1–10

Bottoni P, Koch M, Parisi-Presicce F, Taentzer G (2001) A visualization of OCL using collaborations. In: Gogolla, M, Kobryn, C (eds) "UML" 2001—The unified modeling language, modeling languages, Concepts, and Tools, Springer, pp 257–271. https://doi.org/10.1007/3-540-45441-1_20

Cabot J, Teniente E (2006) metric for measuring the complexity of OCL expressions. In: Models in Software Engineering, Workshops and Symposia—Model Size Metrics Workshop, pp 1–10

Cadavid JJ, Combemale B, Baudry B (2015) An analysis of metamodeling practices for MOF and OCL. Computer Languages Systems & Structures 41:42–65

Casalnuovo C, Devanbu P, Oliveira A, Filkov V, Ray B (2015) Assert use in GitHub projects. In: IEEE International Conference on Software Engineering, IEEE, pp 755–766

Correa A, Werner C, Barros M (2007) An empirical study of the impact of OCL smells and refactorings on the understandability of OCL specifications. In: ACM/IEEE International conference on model driven engineering languages and systems, springer, lecture notes in computer science, vol 4735, pp 76–90

Cuadrado JS, Molina JG (2007) Building domain-specific languages for model-driven development. IEEE Softw 24(5):48–55

Eclipse Juno (2012) Eclipse Juno Documentation. http://help.eclipse.org/juno/index.jsp. Accessed 7 Oct 2015

Ecore (2004) Ecore. http://www.eclipse.org/modeling/emf/. Accessed 20 July 2016

Goeminne M, Mens T (2011) Evidence for the Pareto principle in open source software activity. In: Workshop on Software Quality and Maintainability, pp 74–82

Gogolla M, Cabot J (2016) Continuing a benchmark for UML and OCL design and analysis tools. In: Software technologies: Applications and foundations - collocated workshops, revised selected papers, springer, lecture notes in computer science, vol 9946, pp 289–302

Gogolla M, Büttner F, Richters M (2007) USE: A UML-based specification environment for validating UML and OCL. Sci Comput Program 69(1–3):27–34

Gogolla M, Büttner F, Cabot J (2013) Initiating a benchmark for UML and OCL analysis tools. In: TAP. Springer, Berlin, pp 115–132

Gousios G, Spinellis D (2012) GHTorrent: GitHub's data from a firehose. In: Mining Software Repositories, IEEE, pp 12–21

git (2008) GitHub help — searching code. https://help.github.com/articles/searching-code/. Accessed 14 March 2017

Habela P, Kaczmarski K, Stencel K, Subieta K (2008) OCL As the query language for UML model execution. In: International conference on computational science, springer, lecture notes in computer science, vol 5103, pp 311-320

Hebig R, Ho-Quang T, Chaudron MRV, Robles G, Fernández MA (2016) The quest for open source projects that use UML: mining GitHub. In: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp 173–183

Heitlager I, Kuipers T, Visser J (2007) A practical model for measuring maintainability. In: Machado RJ, Brito e Abreu F, Rupino da Cunha P (eds) Quality of information and communications technology, IEEE pp 30–39

Hermans F, Pinzger M, van Deursen A (2009) Domain-specific languages in practice: A user study on the success factors. In: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, Springer, pp 423–437

Hutchinson JE, Rouncefield M, Whittle J (2011a) Model-driven engineering practices in industry. In: IEEE International Conference on Software Engineering, ACM, pp 633–642

Hutchinson JE, Whittle J, Rouncefield M, Kristoffersen S (2011b) Empirical assessment of MDE in industry. In: IEEE International Conference on Software Engineering, pp 471–480

Jackson D (2012) Software Abstractions: logic, language, and analysis. MIT press, Cambridge

Jouault F, Kurtev I (2006) Transforming models with atl. In: Bruel, J M (ed) Satellite events at the moDELS 2005 conference, lecture notes in computer science, vol 3844, springer, pp 128–138

Kleppe AG, Warmer JB, Bast W (2003) MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional, Boston

Kolovos DS, Paige RF, Polack FAC (2008) The epsilon transformation language. In: Vallecillo, A, Gray, J, Pierantonio, A (eds) International conference on model transformation, springer, pp 46–60

Kolovos DS, Matragkas ND, Korkontzelos I, Ananiadou S, Paige RF (2015) Assessing the use of eclipse MDE technologies in open-source software projects. In: International Workshop on Open Source Software for Model Driven Engineering co-located with ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, pp 20–29

Konietschke F, Hothorn LA, Brunner E (2012) Rank-based multiple test procedures and simultaneous confidence intervals. Electronic Journal of Statistics 6:738–759

Kuhlmann M, Hamann L, Gogolla M (2011) Extensive validation of OCL models by integrating SAT solving into USE In: International conference on objects, models, components, patterns (TOOLS), Springer, pp 290–306

Landman D, Serebrenik A, Vinju JJ (2017) Challenges for static analysis of Java reflection: literature review and empirical study. In: IEEE International Conference on Software Engineering, IEEE / ACM, pp 507–518

Mengerink JGM, Schiffelers RRH, van den Brand MGJ, Serebrenik A (2017a) A case of industrial vs. open-source OCL: Not so different after all. In: ACM/IEEE International Conference on Model Driven Engineering Languages and SystemsSatellite Events, pp 472–474

Mengerink JGM, Serebrenik A, Schiffelers RRH, van den Brand MGJ (2017b) Automated analyses of model-driven artifacts: Obtaining insights into real-life application of MDE. In: 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement (IWSM-Mensura), pp 116–121

Menzies T, Williams l, Zimmermann T (2016) Perspectives on Data Science for Software Engineering. Morgan Kaufmann, Burlington

MOF (1996) MOF. http://www.omg.org/spec/MOF/. Accessed 7 April 2016

Mohagheghi P, Gilani W, Stefanescu A, Fernandez MA (2013) An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. Empir Softw Eng 18(1):89–116

MTL (2016) MTL

Muller J (1998) The great Logo Adventure: Discovering Logo on and off the computer. Doone

Munaiah N, Kroh S, Cabrey C, Nagappan M (2017) Curating github for engineered software projects. Empir Softw Eng 22(6):3219–3253

Murphy-Hill E, Parnin C, Black AP (2012) How we refactor, and how we know it. IEEE Trans Softw Eng 38(1):5–18

Noten J, Mengerink JGM, Serebrenik A (2017a) A data set of OCL expressions on GitHub. In: Mining Software Repositories, pp 531–534

Noten J, Mengerink JGM, Serebrenik A (2017b) A data set of OCL expressions on GitHub. https://doi.org/10.4121/uuid:83317fd5-91f5-4e4b-b475-3e2d8ff12d1c. Accessed: 12 March 2018

Oliveira P, Lima FP, Valente MT, Serebrenik A (2014) RTTool: A tool for extracting relative thresholds for source code metrics. In: IEEE International Conference on Software Maintenance and Evolution, IEEE, pp 629–632

OMG (1989) OMG. http://www.omg.org, accessed: 2017-07-03

Petre M (2013) UML in practice. In: IEEE International Conference on Software Engineering, IEEE, pp 722–731

QVT (2015) QVT. http://www.omg.org/spec/QVT/. Accessed 7 April 2015

QVTo (2015) QVTo. http://www.eclipse.org/mmt/?project=qvto. Accessed 7 April 2015

Reynoso L, Genero M, Piattini M, Manso E (2006) Does object coupling really affect the understanding and modifying of OCL expressions? In: ACM Symposium on Applied Computing, ACM, pp 1721–1727

Richters M, Gogolla M (1998) On formalizing the UML object constraint language OCL. In: Conceptual modeling, springer, lecture notes in computer science, vol 1507 pp 449-464

Robles G, Ho-Quang T, Hebig R, Chaudron MRV, Fernandez MA (2017) An extensive dataset of UML models in GitHub. In: Mining Software Repositories, IEEE, pp 519–522

Romano J, Kromrey JD, Coraggio J, Skowronek J, Devine L (2006) Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen's d indices the most appropriate choices? In: Annual Meeting of the Southern Association for Institutional Research

Rose LM, Kolovos DS, Paige RF, Polack FA (2010) Model migration with Epsilon Flock. In: International Conference on Model Transformation, Lecture Notes in Computer Science, vol 6142, Springer, pp 184–198

Scheidgen M (2006) CMOF-Model semantics and language mapping for MOF 2.0 implementations. In: Joint meeting of the fourth workshop on model-based development of computer-based systems and the third international workshop on model-based methodologies for pervasive and embedded software (MBD/MOMPES), IEEE, pp 84-93

Shull FJ, Carver JC, Vegas S, Juristo N (2008) The role of replications in empirical software engineering. Empir Softw Eng 13:211–218

SysML (2001) OMG SysML. http://www.omgsysml.org/. Accessed 5 July 2016

UML (1997) UML. http://www.uml.org/ accessed: 2016-06-28

Vasilescu B, Serebrenik A, Goeminne M, Mens T (2013) On the variation and specialisation of workload—a case study of the Gnome ecosystem community. Empir Softw Eng 19(4):955–1008

Warmer J, Kleppe A (2003) The Object Constraint Language: Getting Your Models Ready for MDA, 2nd. Addison-Wesley, Boston

Whittle J, Hutchinson JE, Rouncefield M (2014) The state of practice in model-driven engineering. IEEE software 31(3):79–85

Willink ED (2011) Aligning OCL with UML. Electronic Communication of the European Association of Software Science and Technology pp 44

Josh G. M. Mengerink is a doctoral candidate in the Software Engineering Technology group of the Eindhoven University of Technology, where he also obtained his master degree (parsing technology). For his PhD, Josh studies evolution of model-driven domain-specific languages, and maintenance of their related programs (i.e. models). In addition to his doctoral research, Josh is (co-)founder of several software start-ups ranging from information systems to visual analytics.

Jeroen Noten obtained his master degree Computer Science and Engineering at the Eindhoven University of Technology. During his graduation, Jeroen did research into the use of the object constraint language. Nowadays, Jeroen is working as a software developer at ISAAC, an internet service agency in Eindhoven, where he develops web and mobile applications.

Alexander Serebrenik (PhD, K.U. Leuven, Belgium 2003; MSc, Hebrew University, Israel, 1999) is an Associate Professor of software evolution at Eindhoven University of Technology. His research covers a wide range of topics, from source code analysis, to collaborative and human aspects of software engineering. He has co-authored a book "Evolving Software Systems" (Springer Verlag, 2014), and more than 100 scientific papers and articles. He has won Distinguished Paper awards at the International Conference on Software Engineering (2017) and the International Conference on the Quality of Information and Communications Technology (2014). He is the steering committee chair of the International Conference on Software Maintenance and Evolution. He is member of the editorial boards of Empirical Software Engineering (Spinger Verlag), Journal of Systems and Software (Elsevier), and Science of Computer Programming; he has also served on the program committees of such software engineering conferences as ICSE, ESEC/FSE, ICSM(E), MSR, SANER and ICPC, winning several Distinguished Reviewer awards.