Quality Assessment of ATL Model Transformations using Metrics*

M.F. van Amstel and M.G.J. van den Brand

Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
{M.F.v.Amstel|M.G.J.v.d.Brand}@tue.nl

Abstract. One of the key concepts of Model-Driven Engineering (MDE) is model transformations. Because of the crucial role of model transformations in MDE, they have to be treated in a similar way as traditional software artifacts. It is therefore necessary to define and assess their quality. In this paper we present a set of metrics for assessing the quality of ATL model transformations.

1 Introduction

Model-Driven Engineering (MDE) is a software engineering discipline in which models play a central role throughout the entire development process [1]. MDE combines domain-specific languages (DSL) for modeling software systems, and model transformations for transforming models specified in a DSL into equivalent models specified in another language. In this way, models specified in a DSL can be used for various different purposes. Model transformations are in many ways similar to traditional software artifacts, i.e., they have to be used by multiple developers, they have to be maintained according to changing requirements and they should preferably be reused. Therefore, they need to adhere to similar quality standards. To achieve these standards, there should be a methodology for developing model transformations with high quality. A first step in this direction is developing methods for assessing the quality of model transformations. For most other types of software artifacts, e.g., source code and models, there already exist approaches for assessing their quality. The goal of our research is to make the quality of model transformations measurable. In this paper, we focus on model transformations created using the ATL model transformation formalism [2].

Our approach is aimed at assessing the internal quality of a model transformation, i.e., the quality of the model transformation artifact itself [3]. We do this by extracting metrics from the model transformation directly, i.e., we do not consider the input and output models of a model transformation in our

* This work has been carried out as part of the FALCON project under the responsibility of the Embedded Systems Institute with Vanderlande Industries as the industrial partner. This project is partially supported by the Netherlands Ministry of Economic Affairs under the Embedded Systems Institute (BSIK03021) program.

measurements. This is referred to as direct quality assessment [3]. In this paper, we present the metrics we intend to use for assessing the internal quality of ATL model transformations. However, metrics alone are not enough for assessing quality. They need to be related to quality attributes. Therefore, we would like to conduct an empirical study in the same way as described in [4]. However, this is a point for future work and will therefore not be addressed in this paper.

The remainder of this paper is structured as follows. In Section 2, we analyze an ATL model transformation by hand. Section 3 describes the metrics that we have defined for analyzing the quality of model transformations. In Section 4, we describe the tool we created that can be used for automatically extracting metrics from ATL transformations. Section 5 revisits the case we manually analyzed in Section 2 and presents an analysis of another case. In both cases, metrics are automatically extracted from the model transformation. We discuss related work in Section 6. Conclusions and directions for further research are given in Section 7.

2 Example

In this section, we perform an analysis of a simple ATL model transformation by hand. The transformation we chose for our analysis is the book to publication transformation that can be found in the ATL zoo [5]. Its purpose is to transform a book model that adheres to the book metamodel into a publication model that adheres to the publication metamodel. This has to be done as follows. The title of a publication should be the title of a book. The authors attribute of a publication should be set to the concatenation of the authors of each of the chapters separated by the word 'and' and there should be no duplicates. The number of pages of the publication should be the sum of the number of all of the pages of the chapters in the book. The ATL code belonging to the transformation is depicted in Listing 1.1.

2.1 Analysis

The model transformation takes one input model and returns one output model. It consists of one transformation rule. This rule is a matched rule and has no imperative section and has no local variable definitions. Since there is only one rule in this model transformation and it is not abstract, there is no rule inheritance present. This rule has an input pattern consisting of one element and an output pattern also consisting of one element. The input pattern has a condition. This implies that not all model elements in the source model may be transformed. The output pattern has three bindings. One of the bindings is initialized by copying a model element of the source model. The other two bindings are initialized by the results of calls to helpers, i.e., the rule depends on two helpers. There are three helpers defined in the transformation module. These are all the helpers in the transformation, since no other ATL units are imported in the transformation. All three helpers are operation helpers that are defined in a context,

```
module Book2Publication;
create OUT: Publication from IN: Book;

helper context Book!Book def: getAuthors(): String =
  self.chapters
        ->collect(e | e.author)
        ->asSet()
        ->iterate(authorName;
                  acc: String = '' |
                  acc + if   acc = ''
                        then authorName
                        else ' and ' + authorName
                        endif
                 );

helper context Book!Book def: getNbPages(): Integer =
  self.chapters
        ->collect(f | f.nbPages)
        ->iterate(pages;
                  acc: Integer = 0 |
                  acc + pages
                 );

helper context Book!Book def: getSumPages() : Integer =
  self.chapters
        ->collect(f | f.nbPages).sum();

rule Book2Publication {
  from  b   : Book!Book(b.getSumPages() > 2)
  to    out : Publication!Publication(
                title   <- b.title,
                authors <- b.getAuthors(),
                nbPages <- b.getSumPages()
              )
}
```

Listing 1.1. Book to publication transformation

in this case the same context. None of the three operation helpers take arguments. Also, in none of the helpers are local variables defined. All helpers are used to manipulate collections. Two of the three helpers are used in the transformation. The helper `getSumPages()` is called twice and the helper `getAuthors` is called once. All of these calls come from the only transformation rule. The helper `getNbPages()` is never called by a rule or another helper. This may indicate that this helper is obsolete. In the documentation of the transformation is stated that the helpers `getNbPages()` and `getSumPages()` are two different implementations of the same functionality. In the helper `getAuthors()`, a conditional statement is used. This should be taken into consideration when testing the transformation.

For such a small example like this one, it is feasible to do an analysis by hand. However, when model transformations grow larger, it is infeasible to do this. Therefore it is desirable to perform an automated analysis. In Section 3, we describe a set of metrics that can be automatically extracted from ATL transformations by a tool that we describe in Section 4.

3 Metrics

This section describes the metrics we defined for assessing the quality of ATL model transformations. The metrics described here are specific for ATL. However, for most of them a conceptually equivalent metric can be defined for other model transformation formalisms as well.

The metrics can be divided into four categories, viz., rule metrics, helper metrics, dependency metrics, and miscellaneous metrics. In the remainder of this section, we will address each of these categories and elaborate on the metrics belonging to them. An overview of all the metrics can be found in Tables 1, and 2 in Section 5.

3.1 Rule Metrics

A measure for the size of a model transformation is the *number of transformation rules* it encompasses. In ATL, there are different types of rules, viz., matched rules, lazy matched rules, unique lazy matched rules, and called rules. In our metrics, we distinguish between these different types of rules. In case of a completely declarative model transformation, i.e., one with only non-lazy matched rules, it is to be expected that the amount of non-lazy matched rules is related to the size of the input metamodel, since typically a matched rule matches on one metamodel element. However, this is not necessarily the case, since matched rules can have input patterns that match on multiple metamodel elements at the same time or it may be the case that only part of the metamodel needs to be covered by the transformation.

Lazy matched rules and called rules need to be explicitly invoked in an ATL model transformation. Therefore it may be the case that there are lazy matched rules or called rules in an ATL model transformation that are never invoked. This can have a number of reasons, e.g., the rule has been replaced by another rule. To detect such unused rules, we included the metrics *number of unused lazy matched rules* and *number of unused called rules*.

ATL has support for rule inheritance. The use of inheritance may affect the quality of a model transformation in a similar way as it affects object-oriented software [6]. A rule deeper in the rule inheritance tree may be more fault-prone because it inherits a number of properties from its ancestors. Moreover, in deep hierarchies it is often unclear from which rule a new rule should inherit from. To acquire insights into the rule inheritance relations in an ATL model transformation, we defined a number of metrics. We propose to measure the *number of rule inheritance trees* and per such tree the *maximum depth* and the *maximum width*. Furthermore, we defined the metric *number of abstract transformation rules*, and, again, we distinguish matched rules, lazy matched rules, and unique lazy matched rules. We also propose to measure for each abstract rule *the number of children* that inherit from it.

We have also defined a number of metrics on the input and output patterns of rules. The metric *number of input pattern elements* and *number of output pattern elements* measure the size of respectively the input and the output pattern

of rules. Note that since called rules do not have an input pattern, the metric number of input model elements does not include called rules. These two metrics can be combined. The metric *rule complexity change* measures the amount of output model elements that are generated per input model element. For example, if the input pattern consists of one model element and two model elements are generated the rule complexity change is $\frac{2}{1} = 2$. We do not consider model elements that are generated within `distinct foreach` blocks, since the amount of generated elements depends on the input model and can therefore not be determined statically. This metric may be used for measuring the external quality [3] of a model transformation because it addresses the size increase (or decrease) of a model. Note that this metric is defined on matched rules only, since called rules do not have an input pattern. Instead, called rules have parameters similar to operation helpers. Therefore, instead of measuring the number of input patterns, for called rules we measure the *number of parameters per called rule*. It may be the case that some of these parameters are never used for various reasons. To detect this, we propose to measure the *number of unused parameters per called rule*.

The input pattern of a matched rule can be constrained by means of a filter condition. The metric *number of rules with a filter condition* measures the amount of rules that have such an input pattern. Using such filter conditions a rule matches only on a subset of the model elements defined by the input pattern. Therefore it may be the case that there are multiple matched rules that match on the same input model elements. We defined the metric *number of matched rules per input pattern* to measure this. Note that in general ATL does *not* allow multiple rules to match the same input elements, except in the case a rule overrides another rule.

To initialize target model elements, an output pattern has bindings. The metric *number of bindings per output pattern* is another measure for the size of the output pattern of a transformation rule. Typically, the bindings of an output pattern are initialized with attributes and references derived from elements in the input pattern. We propose to measure the *number of unused input pattern elements* to detect possibly obsolete input pattern elements. Matched rules require input pattern elements for the matching. In case that none of the input pattern elements of a lazy matched rule are used in that rule, this could be an indication that a called rule may be used instead. Note, however, that this is not always the case, since switching from a lazy matched rule to a called rule will no longer provide implicit tracing information that can be used elsewhere in the transformation. A metric related to the previous two is the *number of direct copies*. This metric measures the number of rules that copy (part of) an input model element to an output model element without changing any of the attributes. Note that this only occurs when the input metamodel and the output metamodel are the same.

Transformation rules can have local variables. These variables are often used to provide separation of concerns, i.e., to split the calculation of certain output bindings in orderly parts. This should increase the understandability of the rule.

We define two metrics to measure the use of local variables in rules, viz., *number of rules with a using clause* and *number of variables per using clause*. We also measure the *number of unused variables defined in using clauses* to detect obsolete variable definitions.

ATL allows the definition of imperative code in rules in a `do` block. This can be used to perform calculations that do not fit the preferred declarative style of programming. To measure the use of imperative code in a transformation, we defined two metrics, viz., *number of rules with a do section* and *number of statements per do section*.

3.2 Helper Metrics

Besides transformation rules, an ATL transformation also consists of helpers. Therefore the size of a model transformation is also influenced by the *number of helpers* it includes. Two orthogonal distinctions can be made. On the one hand, there are helpers with context and helpers without context. On the other hand, there are attribute helpers and operation helpers. The operation helpers can be further subdivided into operation helpers with parameters and without parameters. Since helpers may be defined in the transformation module or in library units, we also measure the *number of helpers per unit*. This gives an idea of the division of helpers among units.

Similarly to lazy matched rules and called rules, helpers need to be invoked explicitly. Therefore, again, it may be the case there are some helpers present in a model transformation that are never invoked. To detect such unused helpers, we included the metric *number of unused helpers*.

Helpers are identified by their name, context, and, in case of operation helpers, parameters. It is possible to overload helpers, i.e., define helpers with the same name but with a different context. To measure this kind of overloading we define the metrics *number of overloaded helpers* and *number of helpers per helper name*. Overloading is used to define similar operations on different datatypes. Of course it is also possible to define multiple different operations on the same datatype, i.e., in a different context. Therefore, we propose to measure the *number of helpers per context*.

To get more insight in the use of parameters by operation helpers, we propose to measure the *number of parameters per operation helper*. Also parameters may be unused for various reasons. To detect this, we propose the metric *number of unused parameters per operation helper*.

Helpers are often used to manipulate collections. Therefore, we measure the *number of operations on collections per helper*. Also, conditions are often used in helpers. The metric *helper cyclomatic complexity* is related to McCabe's cyclomatic complexity [7], it measures the amount of decision points in a helper. Currently, only `if` statements are considered as decision points. Similar to rules, helpers also allow the definition of local variables. We define the metrics *number of helpers with a let clause*, and *number of variables per helper* to measure the use of variables in helpers. Again, we also measure the *number of unused variables defined in let clauses* to detect obsolete variable definitions.

3.3 Dependency Metrics

We consider six categories of dependency, viz., units depending on other units, rules depending on rules, rules depending on helpers, helpers depending on helpers, helpers depending on called rules, and rules and helpers on built-in functions. To measure the first three categories of dependencies we have defined a number of *fan-in*, and *fan-out* metrics. For example, the metric *number of calls to lazy matched rules per lazy matched rule* measures the fan-in of lazy matched rules and the metric *number of calls from helpers to helpers per helper* measures the fan-out of helpers. An overview of all the fan-in and fan-out metrics can be found in Table 2 in Section 2.

The dependency of units on other units is measured by four metrics. The metrics *number of imported units per unit* and *number of times a module is imported per unit* are used to measure the import dependencies of units. On a lower level, the metrics *number of calls from helpers in other units (unit fan-in)* and *number of calls to helpers in other units (unit fan-out)* measure how the internals of units depend on each other.

The last dependency category, i.e., the dependency of rules and helpers on built-in functions is measured by the metric *number of calls to built-in functions*. Built-in functions are OCL functions and also additional ATL operations such as `replaceAll()`. There are some built-in functions that deserve special attention. First, there is the `resolveTemp()` function. This function is used to look-up references to non-default output elements of other rules. Therefore, it is to be expected that model transformations with a large number of calls to the `resolveTemp()` function are harder to understand. There are also the `println()` and the `debug()` function. The `debug()` function is used to print information to the console that can be used for debugging. In practice, we see that sometimes the `println()` function is used for a similar purpose. The occurrence of these two functions may indicate that the model transformation is still under development.

3.4 Miscellaneous Metrics

We defined four more metrics that do not fit the discussed categories. The metric *number of units* measures the amount of units that make up a model transformation. This metric can provide insight in the size and modularity of a model transformation. It may be the case that there are library units imported from which no helper is invoked in the transformation. To detect this, we define the metric *number of unused units*.

The last two metrics provide insight in the context of the model transformation. It is to be expected that model transformations involving more models are more complex. Therefore we propose to measure the *number of input models* and the *number of output models*.

4 Tool

We implemented a tool that enables automatic collection of the metrics presented in Section 3 from ATL specifications [8]. Figure 1 gives an overview of the architecture of the tool. An ATL model transformation consists of a mod-

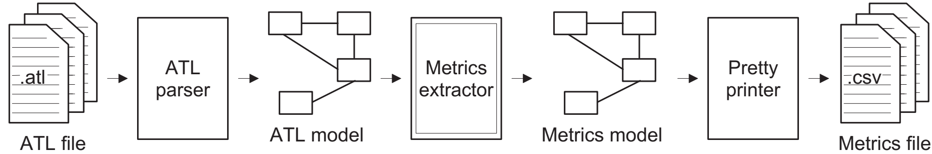

Fig. 1. Metrics extraction tool architecture

ule and possibly a number of libraries. The files containing this module and libraries are parsed by the ATL parser. This results in ATL models representing the model transformation. These models are the input for the metrics extractor. This metrics extractor is itself a model transformation implemented in ATL. It consists of one matched rule that matches on an ATL module, and a number of lazy matched rules, each for calculating the value of one of the metrics. The output of the metrics extractor is a model that contains the metrics data. This model is used as input for a pretty printer. This pretty printer is a model-to-text transformation implemented in Xpand [9]. The output of the pretty printer is a comma separated value file that can be read by a spreadsheet application.

In Section 3 we presented two types of metrics, viz., metrics that are measured over the entire transformation and metrics that are measured on a smaller scale, i.e., per unit, per rule, or per helper. The former type of metric has as single value for the entire transformation. We refer to this type of metrics as *simple metrics*. The latter type of metric has multiple values for the entire transformation, viz., one for every element that is measured (unit, rule, or helper). To assess the transformation as a whole we do not present all of these values. Instead, we give average, minimum, maximum, median and standard deviations for these metrics. We refer to this type of metrics as *aggregated metrics*.

The metrics extractor gathers data from an ATL model. Therefore, there is a problem with identifying calls to helpers. The call to a helper consists of only its name. However, a helper is defined by its name, context, and parameters. In order to distinguish between calls to helpers with the same name, more information is required. Unfortunately, this information is only available during run-time. We therefore decided to identify a call to a helper only by its name. We realize that this is a threat to the validity of our measurements, since some of the metrics may be calculated incorrectly. However, from our own experience and by looking at the transformations available in the ATL zoo [5] we can conclude that there are few transformations that have overloaded helpers. Besides, if there are overloaded helpers, this is indicated by a metric.

5 Example: Continued

In Section 2, we performed a manual analysis of the book to publication transformation. In this section, we present the metric values that were automatically extracted from this transformation by our metrics extraction tool. Table 1 contains the simple metrics and Table 2 contains the aggregated metrics. The first column of the tables states the category a metric belongs to. The second column contains the metric itself. The remainder of the columns contain the values for a metric.

	Metric	Value
Rule metrics	# Transformation Rules	1
	# Matched Rules (Excluding Lazy Matched Rules)	1
	# Matched Rules (Including Lazy Matched Rules)	1
	# Lazy Matched Rules (Excluding Unique)	0
	# Lazy Matched Rules (Including Unique)	0
	# Unique Lazy Matched Rules	0
	# Called Rules	0
	# Unused Lazy Matched Rules	0
	# Unused Called Rules	0
	# Rules with a Filter Condition on an Input Pattern	1
	# Unused Input Pattern Elements	0
	# Rules with a do Section	0
	# Direct Copies	0
	# Abstract Transformation Rules	0
	# Abstract Matched Rules	0
	# Abstract Lazy Matched Rules	0
	# Abstract Unique Lazy Matched Rules	0
	# Rule Inheritance Trees	1
	# Rules with a Using Clause	0
	# Unused Variables Defined in Using Clauses	0
Helper metrics	# Helpers	3
	# Helpers with Context	3
	# Helpers without Context	0
	# Attribute Helpers	0
	# Attribute Helpers with Context	0
	# Attribute Helpers without Context	0
	# Operation Helpers	3
	# Operation Helpers with Context	3
	# Operation Helpers without Context	0
	# Operation Helpers with Parameters	0
	# Operation Helpers without Parameters	3
	# Overloaded Helpers	0
	# Unused Helpers	1
	# Unused Helpers with Context	1
	# Unused Helpers without Context	0
	# Unused Attribute Helpers	0
	# Unused Operation Helpers	1
	# Helpers with a Let Clause	0
	# Unused Variables Defined in Let Clauses	0
Dep.	# Calls to `println()`	0
	# Calls to `debug()`	0
	# Calls to `resolveTemp()`	0
	# Calls To Built-In Functions (Built-In Function Fan-In)	7
Misc.	# Units	1
	# Unused Units	0
	# Input Models	1
	# Output Models	1

Table 1. Metrics extracted from the book to publication transformation (1)

	Metric	Avg.	Min.	Max.	Med.	StdDev.
Rule metrics	# Elements per Input Pattern	1.0	1	1	1	0.0
	# Elements per Output Pattern	1.0	1	1	1	0.0
	# Matched Rules per Input Pattern	1.0	1	1	1	0.0
	# Parameters per Called Rule	0.0	0	0	0	0.0
	# Unused Parameters per Called Rule	0.0	0	0	0	0.0
	# Variables per Using Clause	0.0	0	0	0	0.0
	# Statements per do Section	0.0	0	0	0	0.0
	# Children per Matched Rule	0.0	0	0	0	0.0
	Depth of Rule Inheritance Tree	1.0	1	1	1	0.0
	Width of Rule Inheritance Tree	1.0	1	1	1	0.0
	# Bindings per Rule	3.0	3	3	3	0.0
	Rule Complexity Increase	1.0	1.0	1.0	1.0	0.0
Helper metrics	# Helpers per Unit	3.0	3	3	3	0.0
	# Attribute Helpers per Unit	0.0	0	0	0	0.0
	# Operation Helpers per Unit	3.0	3	3	3	0.0
	# Helpers per Context	3.0	3	3	3	0.0
	# Helpers per Helper Name (overloadings)	1.0	1	1	1	0.0
	Helper Cyclomatic Complexity	1.33	1	2	1	0.58
	# Operations on Collections per Helper	1.33	1	2	1	0.58
	# Variables per Helper	0.0	0	0	0	0.0
	# Parameters per Operation Helper	0.0	0	0	0	0.0
	# Unused Parameters per Operation Helper	0.0	0	0	0	0.0
Dependency metrics	# Imported Units	0.0	0	0	0	0.0
	# Times a Unit is Imported	0.0	0	0	0	0.0
	# Calls to Lazy Matched Rules per Lazy Matched Rule (Fan-In)	0.0	0	0	0	0.0
	# Calls to Called Rules per Called Rule (Fan-In)	0.0	0	0	0	0.0
	# Calls to Helpers per Helper (Fan-In)	1.0	0	2	1	1.0
	# Calls to Attribute Helpers per Attribute Helper (Fan-In)	0.0	0	0	0	0.0
	# Calls to Operation Helpers per Operation Helper (Fan-In)	1.0	0	2	1	1.0
	# Calls from Helpers to Called Rules per Helper (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Helpers to Helpers per Helper (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Helpers to Attribute Helpers per Helper (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Helpers to Operation Helpers per Helper (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Rules to Helpers per Rule (Fan-Out)	3.0	3	3	3	0.0
	# Calls from Rules to Rules per Rule (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Rules to Called Rules per Rule (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Rules to Lazy Matched Rules per Rule (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Helpers in Other Units (Unit Fan-In)	0.0	0	0	0	0.0
	# Calls to Helpers in Other Units (Unit Fan-Out)	0.0	0	0	0	0.0
	# Calls to `resolveTemp()` per Rule	0.0	0	0	0	0.0

Table 2. Metrics extracted from the book to publication transformation (2)

As a second example, we present the metrics we extracted from the metrics extractor described in Section 4 using that same tool. The extracted metrics are presented in Tables 3 and 4.

	Metric	Value
Rule metrics	# Transformation Rules	90
	# Matched Rules (Excluding Lazy Matched Rules)	1
	# Matched Rules (Including Lazy Matched Rules)	90
	# Lazy Matched Rules (Excluding Unique)	89
	# Lazy Matched Rules (Including Unique)	89
	# Unique Lazy Matched Rules	0
	# Called Rules	0
	# Unused Lazy Matched Rules	1
	# Unused Called Rules	0
	# Rules with a Filter Condition on the Input Pattern	0
	# Unused Input Pattern Elements	83
	# Rules with a do Section	0
	# Direct Copies	0
	# Abstract Transformation Rules	0
	# Abstract Matched Rules	0
	# Abstract Lazy Matched Rules	0
	# Abstract Unique Lazy Matched Rules	0
	# Rule Inheritance Trees	90
	# Rules with a Using Clause	55
	# Unused Variables Defined in Using Clauses	0
Helper metrics	# Helpers	28
	# Helpers with Context	11
	# Helpers without Context	17
	# Attribute Helpers	1
	# Attribute Helpers with Context	0
	# Attribute Helpers without Context	1
	# Operation Helpers	27
	# Operation Helpers with Context	11
	# Operation Helpers without Context	16
	# Operation Helpers with Parameters	13
	# Operation Helpers without Parameters	14
	# Overloaded Helpers	0
	# Unused Helpers	0
	# Unused Helpers with Context	0
	# Unused Helpers without Context	0
	# Unused Attribute Helpers	0
	# Unused Operation Helpers	0
	# Helpers with a Let Clause	14
	# Unused Variables Defined in Let Clauses	0
Dep.	# Calls to println()	7
	# Calls to debug()	0
	# Calls to resolveTemp()	0
	# Calls To Built-In Functions (Built-In Function Fan-In)	519
Misc.	# Units	1
	# Unused Units	0
	# Input Models	1
	# Output Models	1

Table 3. Metric extracted from the ATL to metrics transformation (1)

	Metric	Avg.	Min.	Max.	Med.	StdDev.
Rule metrics	# Elements per Input Pattern	1.0	1	1	1	0.0
	# Elements per Output Pattern	1.0	1	1	1	0.0
	# Matched Rules per Input Pattern	90.0	90	90	90	0.0
	# Parameters per Called Rule	0.0	0	0	0	0.0
	# Unused Parameters per Called Rule	0.0	0	0	0	0.0
	# Variables per Using Clause	2.22	1	6	2	1.17
	# Statements per do Section	0.0	0	0	0	0.0
	# Children per Matched Rule	0.0	0	0	0	0.0
	Depth of Rule Inheritance Tree	1.0	1	1	1	0.0
	Width of Rule Inheritance Tree	1.0	1	1	1	0.0
	# Bindings per Rule	3.84	2	6	2	1.99
	Rule Complexity Increase	1.0	1.0	1.0	1.0	0.0
Helper metrics	# Helpers per Unit	28.0	28	28	28	0.0
	# Attribute Helpers per Unit	1.0	1	1	1	0.0
	# Operation Helpers per Unit	27.0	27	27	27	0.0
	# Helpers per Context	2.75	1	7	1	2.88
	# Helpers per Helper Name (Overloadings)	1.0	1	1	1	0.0
	Helper Cyclomatic Complexity	2.07	1	7	2	1.18
	# Operations on Collections per Helper	3.39	1	7	3	1.54
	# Variables per Helper	0.5	0	1	0	0.51
	# Parameters per Operation Helper	0.52	0	2	0	0.58
	# Unused Parameters per Operation Helper	0.0	0	0	0	0.0
Dependency metrics	# Imported Units	0.0	0	0	0	0.0
	# Times a Unit is Imported	0.0	0	0	0	0.0
	# Calls to Lazy Matched Rules per Lazy Matched Rule (Fan-In)	0.99	0	1	1	0.11
	# Calls to Called Rules per Called Rule (Fan-In)	0.0	0	0	0	0.0
	# Calls to Helpers per Helper (Fan-In)	12.14	1	48	3	16.40
	# Calls to Attribute Helpers per Attribute Helper (Fan-In)	1.0	1	1	1	0.0
	# Calls to Operation Helpers per Operation Helper (Fan-In)	12.56	1	48	4	16.56
	# Calls from Helpers to Called Rules per Helper (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Helpers to Helpers per Helper (Fan-Out)	0.64	0	4	0	0.99
	# Calls from Helpers to Attribute Helpers per Helper (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Helpers to Operation Helpers per Helper (Fan-Out)	0.64	0	4	0	0.99
	# Calls from Rules to Helpers per Rule (Fan-Out)	3.58	0	12	1	3.36
	# Calls from Rules to Rules per Rule (Fan-Out)	0.97	0	87	0	9.17
	# Calls from Rules to Called Rules per Rule (Fan-Out)	0.0	0	0	0	0.0
	# Calls from Rules to Lazy Matched Rules per Rule (Fan-Out)	0.97	0	87	0	9.17
	# Calls from Helpers in Other Units (Unit Fan-In)	0.0	0	0	0	0.0
	# Calls to Helpers in Other Units (Unit Fan-Out)	0.0	0	0	0	0.0
	# Calls to resolveTemp() per Rule	0.0	0	0	0	0.0

Table 4. Metric extracted from the ATL to metrics transformation (2)

The metric values give some insight into the transformation. There are 89 lazy matched rules and it can be derived from the metric values that all matched rules have one element per input pattern. Also, 83 input pattern elements are unused. From this we can conclude that 83 of the matched rules do not use their input pattern. This indicates that a large part of the lazy matched rules may be replaced by called rules if the use of the implicit tracing mechanism is not required. In this particular case it is possible to replace lazy matched rules by called rules. This is a point for future work.

There are seven calls to the `println()` function. In Section 3, we argued that this could indicate that debugging information is being printed. Here this is not the case. Instead, the calls to the `println()` function are used to generate warnings.

There are 90 rules and 28 helpers in one module. It may be advisable to introduce a library for the helpers to decrease the size of the transformation module.

From the values for the metric *number of calls to helpers per helper*, we can conclude that there are a lot of calls to a few helpers. These are calls to the helpers that calculate the aggregated metrics.

There is one unused lazy matched rule. This lazy rule is intended to calculate a metric that we considered as irrelevant. It should therefore be removed.

The advantages of using the tool to extract metric values are that it is fast, repeatable, and less error-prone. The disadvantage of using the tool is that it only provides numbers. Therefore, the analysis is not as detailed as the manual analysis we presented in Section 2. For example, from the metrics we can only conclude that there is one unused helper in the transformation, not which one it is. This is not a problem since we are interested in a relation between metrics and quality attributes. Moreover, the tool can easily be extended to provide more detailed reports such that it can be of assistance in locating problems.

6 Related Work

In [10], a similar study aimed at defining metrics for ATL model transformations is described. We used the metrics they defined to complete our own set of metrics. Therefore, the metrics they define overlap with ours. Some of the metrics they consider are used to measure the input and output metamodels. Therefore they perform both direct and indirect quality assessment, whereas we consider direct quality assessment only. In our metrics set, there are more metrics aimed at detecting incompleteness and inconsistency, e.g., number of unused lazy matched rules, unused units, and number of calls to `debug()`. Our goal is to establish a relation between metrics and quality attributes by means of an empirical study. They sketch such a relation for their metrics based on how they expect it to be.

The authors of [11] discuss characteristics of MDE that are important when building a quality framework for it. One of the characteristics that needs to be considered is the quality of model transformations. Their motivation for this is that model transformations can affect the quality of models. Another reason for considering the quality of model transformations they present is that model transformations are increasingly being used at runtime in self-adaptive systems where performance plays an important role. They discuss some issues that play a role in model transformation quality and they fit it into their framework. The main difference with our work is the scope. They consider the quality of different aspects of MDE, whereas our focus is solely on model transformation quality. Also their scope on model transformation quality is broader. They studied literature and identified issues regarding the evaluation of model transformation quality and fit that in their framework, whereas we propose a methodology for assessing the internal quality of model transformations using metrics.

The authors of [12] state that one of the problems in MDE is how to identify what model transformations can improve the quality if models. They propose

to solve this problem by embedding metrics and methods for calculating them into metamodels. The values for these metrics can be calculated before and after performing the model transformation to establish how the quality of the model has changed. The main difference with our approach is that they focus on the external quality of model transformations, whereas we focus on their internal quality. The external quality of a model transformation is the quality change induced on a model by the model transformation [3]. The disadvantage of their approach is that it only applies to endogenous model transformations, i.e., model transformations where the input and output model adhere to the same metamodel [13]. In case the input and output model do not adhere to the same metamodel, i.e., in case of an exogenous model transformation, the metrics for measuring model quality are probably different and therefore incomparable.

7 Conclusions and Future Work

We have addressed the necessity for a methodology to analyze the quality of model transformations. In this paper, we defined a set of 86 metrics for predicting the quality of model transformations created using ATL. These metrics can automatically be collected from ATL model transformation specifications by a tool we created.

Metrics alone are not sufficient to assess the quality of a model transformation. They need to be related to quality attributes to understand which metrics are relevant for assessing the different attributes of the quality model of transformations. Therefore, we would like to conduct an empirical study in the same way as described in [4]. Metrics should be extracted from a heterogeneous collection of ATL model transformations using the tool described in Section 4. The quality of the same collection of model transformations should be manually assessed by ATL experts. Thereafter the correlations between the metrics data acquired from the tool and the quality evaluation acquired from the experts should be analyzed. In this way, we can derive which metrics are relevant for assessing the different attributes of model transformation quality.

Acknowledgement

We would like to thank Andrés Vignaga for providing us with and letting us use the metrics he defined in [10]. We would also like to thank the anonymous reviewers for their valuable comments which helped us improving an earlier version of this paper.

References

1. Schmidt, D.C.: Model-driven engineering. Computer **39**(2) (February 2006) 25 – 31

2. Jouault, F., Kurtev, I.: Transforming models with ATL. In Bruel, J., ed.: MoDELS 2005 Satellite Events. Number 3844 in Lecture Notes in Computer Science, Montego Bay, Jamaica, Springer (October 2005) 128 – 138
3. van Amstel, M.F.: The right tool for the right job: Assessing model transformation quality. In: Proceedings of the Fourth IEEE International Workshop on Quality Oriented Reuse of Software (QUORS'10) (co-located with COMPSAC 2010), Seoul, South-Korea (July 2010) To appear.
4. van Amstel, M.F., Lange, C.F.J., van den Brand, M.G.J.: Using metrics for assessing the quality of ASF+SDF model transformations. In Paige, R.F., ed.: Proceedings. of the Second International Conference on Model Transformation. Volume 5563 of Lecture Notes in Computer Science., Zürich, Switzerland, Springer (June 2009) 239 – 248
5. ATL transformations.
http://www.eclipse.org/m2m/atl/atlTransformations/ (viewed March 2010)
6. Basili, V.R., Briand, L.C., Melo, W.L.: A validation of object-oriented design metrics as quality indicators. IEEE Transactions on Software Engineering **22**(10) (1996) 751 – 761
7. McCabe, T.J.: A complexity measure. IEEE Transactions on Software Engineering **2**(4) (December 1976) 308 – 320
8. van Amstel, M.F.: Metrics extraction tool.
http://www.win.tue.nl/~mamstel/experiments.html#ATL (2010)
9. Thoms, K., Efftinge, S.: Xpand website.
http://wiki.eclipse.org/Xpand (viewed March 2010)
10. Vignaga, A.: Metrics for measuring ATL model transformations. Technical report, MaTE, Department of Computer Science, Universidad de Chile (2009)
11. Mohagheghi, P., Dehlen, V.: An overview of quality frameworks in model-driven engineering and observations on transformation quality. In: Proceedings of the Second Workshop on Quality in Modeling (QiM'07), Nashville, USA (October 2007)
12. Saeki, M., Kaiya, H.: Measuring model transformation in model driven development. In Eder, J., Tomassen, S.L., Opdahl, A.L., Sindre, G., eds.: Proceedings of the CAiSE'07 Forum at the 19th International Conference on Advanced Information Systems Engineering. Volume 247 of CEUR Workshop Proceedings., Trondheim, Norway, CEUR-WS.org (June 2007)
13. Mens, T., van Gorp, P.: A taxonomy of model transformation. In Karsai, G., Taentzer, G., eds.: Proceedings of the International Workshop on Graph and Model Transformation (GraMoT'05). Volume 152 of Electronic Notes in Theoretical Computer Science., Tallinn, Estonia, Elsevier (September 2006) 125 – 142