Metrics for Measuring ATL Model Transformations*

Andrés Vignaga

MaTE, Department of Computer Science, Universidad de Chile
avignaga@dcc.uchile.cl

Abstract

ATL is currently one of the most popular model transformations languages. However, so far the quality of transformations defined in ATL has received little attention. Metrics is a widely used approach for assessing the quality of software, and was already applied in the context of other model transformation languages. In this work we present a set of metrics which make ATL transformations measurable, and enables assessing their quality.

1 Introduction

Model transformations are key assets in Model Driven Engineering (MDE). They embody the primary means for manipulating (e.g., creating or modifying) models, which are first class constructs within MDE. In particular, an MDE-based process may be partially carried out by executing a series of different model transformations on the artifacts associated to that process. As a consequence, the quality of model transformations directly affects the practical application of MDE. Assessing the quality of model transformations provides valuable information for managing and controlling both model transformation development and the process in which they are applied. Software metrics are a widely used method for assessing the quality of software in general, and have also been applied to model transformations.

The AtlanMod Transformation Language (ATL) [4] is currently one of the most popular model transformation language within the MDE community and industry. A set of metrics which specifically apply to measuring model transformations defined in ATL is not yet defined. In this paper we address this particular issue.

Quality issues related to ATL transformations were addressed in the *ATL2Problem* use case [1]. *ATL2Problem* is an ATL transformation that checks a set of non-structural constraints on the ATL definition of concrete transformations, such as rule name uniqueness. As a result, a Problem model is produced where a Problem element represents a constraint violation. If regarded as metrics, each kind of problem then may take a binary value: present or not present. Model transformation metrics have been addressed in [8]. There, a quality model which defines the meaning of quality attributes in the context of model transformation is proposed. However, metrics for assessing those quality attributes specifically target transformations defined using the ASF+SDF term rewriting system. In turn, a framework for measuring model transformations was proposed in [7]. In such a framework, measures do not refer to a transformation definition. Rather, model transformation quality is indicated by the measure of how much a transformation improves its involved models. To this end, model-specific metrics are computed on source and target models of the transformation to be measured, and the improvement is indicated by calculating the difference or norm of such measures.

*This work was carried out in collaboration with the AtlanMod team (INRIA and EMN), and was funded by CONICYT Chile and Universidad de Chile.

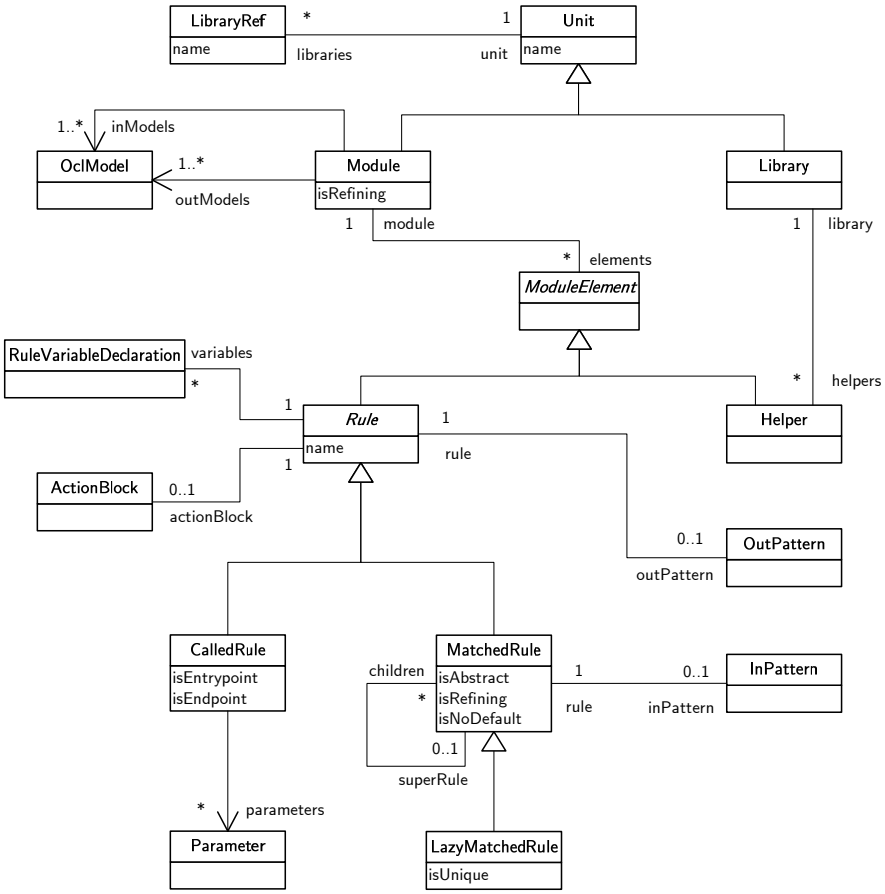

Figure 1: Partial ATL metamodel (part 1)

Based on the quality attributes introduced in [8] for model transformations, in this work we define a set of of metrics which can be applied for measuring model transformations defined in ATL. The measures obtained from applying those metrics to a concrete model transformation enable assessing those quality attributes. Our approach may be applied to individual ATL files. However, more generally, we also consider *complete* transformation definitions in ATL. That is, for us, a complete ATL transformation is composed of a module and the transitive closure of all imported libraries.

The rest of this work is structured as follows. Section 2 introduces the basic constructs of ATL transformations for providing a foundation for metric definition. Section 3 enumerates the quality attributes to be considered, and presents the set of metrics for ATL transformations. In Sect. 4 we discuss the relation of the metrics to the quality attributes. Section 5 concludes and discusses further work.

2 AtlanMod Transformation Language

In this work we focus on metrics for measuring model transformations defined in ATL. In this section we summarize the main constructs of the language in order to provide a foundation for metric definition. ATL is a hybrid transformation language. It enables the definition of a model transformation in a declarative, rule-based style, in an imperative style, or both. However, the declarative style should be preferred [3]. A simplification of the core part of the ATL metamodel [2] is shown in Fig. 1. A transformation is

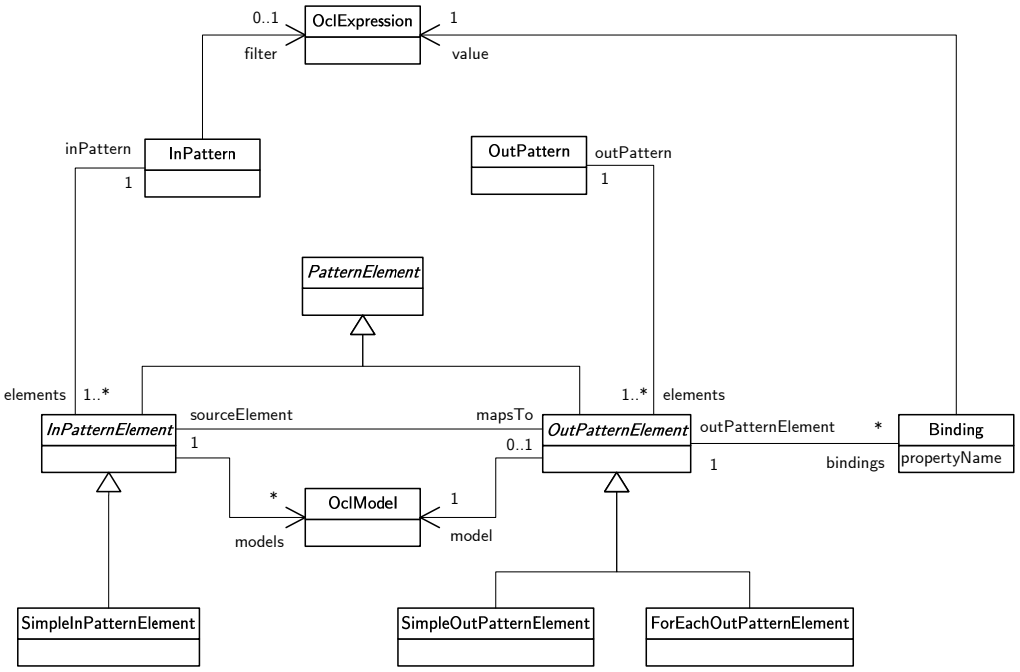

Figure 2: Partial ATL metamodel (part 2)

defined within a Module and operates on a non-empty set of source models (inModels) for producing a non-empty set of target models (outModels). A Module has a number of associated ModuleElements which can be either Rules or Helpers. A Helper is a query function, which is essentially an OCL expression. It is usually defined in the context of a model element within a source metamodel (target models are write-only), or in the context of the Module itself. A special form of Helper, with no parameters, is called an *attribute*. For efficiency reasons, a Helper with no parameters should be defined as an attribute. A Helper may be used either from another Helper or from a Rule.

ATL provides two kinds of Rules: MatchedRule and CalledRule. They correspond to the declarative and imperative styles respectively. Note that both a Module may have Rules of both kinds at the same time. A Rule may define local variables, and may have an ActionBlock of imperative code which is to be executed when the Rule is triggered.

A MatchedRule defines a mapping between a set of source elements (specified by its InPattern) and a set of target elements (specified by its OutPattern). The details of these two elements is shown in Fig. 2. An InPattern is composed of a set of InPatternElements, which are to be automatically matched to source elements within a source model. When a match is got, the Rule triggers, if the optional filter condition is satisfied. As a result, each OutPatternElement within the OutPattern is created in a target model. Properties of these target elements are initialized by the corresponding Binding using the specified value. Elements of an OutPattern may generate individual model elements (SimpleOutPatternElement) or a collection of model elements of arbitrary length (ForEachOutPatternElement). A LazyMatchedRule is a special kind of MachtedRule. The only difference is that it is not automatically matched; it is explicitly called instead. Finally, a unique LazyMatchedRule uses the same created target elements across all its calls.

A CalledRule may be regarded as a sort of helper which produces target elements (a helper may return data values only). A CalledRule needs to be explicitly called from imperative code. The only exception is the case of a CalledRule designated as entrypoint,

```
module Families2Persons;
create OUT : Persons from IN : Families;

helper context Families!Member def: familyName : String =
        if not self.familyFather.oclIsUndefined() then
                self.familyFather.lastName
        else
                if not self.familyMother.oclIsUndefined() then
                        self.familyMother.lastName
                else
                        if not self.familySon.oclIsUndefined() then
                                self.familySon.lastName
                        else
                                self.familyDaughter.lastName
                        endif
                endif
        endif;

helper context Families!Member def: isFemale() : Boolean =
        if not self.familyMother.oclIsUndefined() then
                true
        else
                if not self.familyDaughter.oclIsUndefined() then
                        true
                else
                        false
                endif
        endif;

rule Member2Male {
        from
                s : Families!Member (not s.isFemale())
        to
                t : Persons!Male (
                        fullName <- s.firstName + ' ' + s.familyName
                )
}

rule Member2Female {
        from
                s : Families!Member (s.isFemale())
        to
                t : Persons!Female (
                        fullName <- s.firstName + ' ' + s.familyName
                )
}
```

Figure 3: ATL source code of transformation *Families2Persons*

which is automatically called when the transformation begins its operation. A CalledRule accepts parameters and does not have an InPattern. In turn, unlike a MatchedRule, the OutPattern is optional for a CalledRule.

A Library is a special kind of Unit which includes a set of Helpers. These Helpers may be called from any Unit (a Module or even another Library) that imports the Library which includes them. Unlike a Module, a Library cannot be executed on its own, and is thus a means for modularizing functionality which may be reused in several contexts.

Every Module has a special variable thisModule which represents the Module itself. This variable may be used for calling CalledRules, LazyMatchedRules, Helpers defined in the context of the Module, and also a series of other operations. One of them is the resolveTemp() operation. This operation is usually called from a Binding of a MatchedRule, or from a Helper which is called from that same location. It may be used for accessing any of the target model elements which *will* be generated from a specific source model element by a MatchedRule. This operation enables a powerful control over target elements, but its use demands extreme care. For example, since Rules are matched in arbitrary order, when target model elements need to be ordered in a specific fashion, resolveTemp() may be used for arranging model elements in an intermediate structure before they are added into a target model, which is unchangeable. We refer the reader to [10] for an example of an intensive use of resolveTemp().

As an example of a concrete ATL module, Fig. 3 shows the source code of a simple transformation from the ATL Transformation Zoo [1]. The module has no library references, and the in and out models are Families and Persons respectively. *Families2Persons* has four elements: two helpers and two rules. Helper isFemale is an operation with no parameters defined in the context of metaclass Member from metamodel Families, while familyName is actually an attribute defined for Member. Both rules are matched rules, and have an in and out pattern, but have no variables or action block. Since they are structurally similar we discuss Member2Male only. The in pattern is what is found after the from keyword, while the out pattern is what is found after the to keyword. The in pattern of Member2Male has a filter which is the OCL expression not s.isFemale(), and just one element. Such an element is necessarily an instance of class SimpleInPatternElement. Although not specified in Fig. 2, PatternElement is a subclass of VariableDeclaration. For the element of the in pattern of Member2Male, the values for the inherited varName and type attributes are s and Member respectively (the associated model is Families). The out pattern of Member2Male has one element of class SimpleOutPatternElement, which is naturally associated to the element of the in pattern just discussed. The element is also a variable declaration, and the values of varName and type in this case are t and Male respectively (the associated model is Persons). This element has just one associated binding. The propertyName is fullName, which is intended to be a property of t, and the value used for its initialization is the OCL expression s.firstName + ' ' + s.familyName.

3 Metrics

In this section a number of metrics for ATL model transformations is presented. They are organized according to the ATL metaclass to which they apply. In the case a metric applies to every variant of the same ATL element, the metric is associated to that element even if in the ATL metamodel it is an abstract metaclass. First, we start by discussing the quality attributes we intend to assess with those metrics.

3.1 Quality Attributes

A number of quality attributes, many of which apply to software in general as well, where identified as relevant for model transformations. Such quality attributes are: Understandability, Modifiability, Reusability, Reuse, Modularity, Completeness, Consistency and Conciseness. The names of attributes should be self explanatory, however a more detailed description of what they mean in the context of model transformations may be found in [8]. In addition to those quality attributes, we also consider Complexity and Performance which were also identified as relevant in the quality model presented in [5].

3.2 Unit Metrics

ATL units considered in this document are Modules and Libraries. ATL Queries are not taken into consideration. Metrics that apply to each variant of Unit include:

- **NIL (Number of Imported Libraries)**
  This metric counts the number of libraries which are directly imported by an ATL unit.
- **TIL (Total number of Imported Libraries)**
  This metric counts the total number of imported libraries, whether directly or indirectly, by an ATL unit. It is the transitive closure of the previous metric.
- **NH (Number of Helpers)**
  This metric counts the number of helpers that are directly defined in an ATL unit; helpers defined in imported libraries are not counted.

- **NHP (Number of Helpers without Parameters)**
  This metric counts the total number of helpers which do not have formal parameters.

- **BOU (Balance Of a Unit)**
  This metric calculates the quotient between the NH measure of a unit and the average of the NH measures of all units.

- **NHC (Number of Helpers per Context)**
  This metric counts the number of helpers that are defined for a given context, that is, a metaclass of a source metamodel, a primitive type, or in the case of a module, the module itself.

- **NOH (Number of Overloaded Helpers)**
  This metric counts the number of helpers whose contexts are different but their names are equal.

3.2.1 Module Metrics

The following metrics specifically apply to ATL Modules:

- **NSM (Number of Source Models)**
  This metric counts the number of source models of a module. Every source model is counted, even if more than one of them conforms to the same metamodel.

- **NTM (Number of Target Models)**
  This metric counts the number of target models of a module. Every target model is counted, even if more than one of them conforms to the same metamodel.

- **ASSM (Average Size of Source Metamodels)**
  This metric calculates the average of the sizes of all source metamodels of a module. The size of one metamodel can be calculated using (a combination of) metrics for metamodels such as those used in [9].

- **ASTM (Average Size of Target Metamodels)**
  This metric calculates the average of the sizes of all target metamodels of a module.

- **CSM (Coverage of Source Metamodels)**
  This metric calculates the quotient between the total number of distinct metaclasses from all source metamodels whose instances are used in a transformation for producing the target models, and the total number of metaclasses from all source metamodels.

- **CTM (Coverage of Target Metamodels)**
  This metric calculates the quotient between the total number of distinct metaclasses from all target models whose instances conform any of the target models, and the total number of metaclasses from all target metamodels.

- **NA (Number of Attributes)**
  This metric counts the total number of attribute helpers that are defined in a module.

- **TA (Transformation Approach)**
  This metric determines the approach followed by a transformation. Its possible values are:

  - Declarative: when all rules are matched rules and none of them has an imperative block section

  - Hybrid: otherwise

- **NMR (Number of Matched Rules)**
  This metric counts the number of matched rules defined in the module.

- **NLR (Number of Lazy matched Rules)**
  This metric counts the number of matched rules in a module which are specifically lazy.

- **NUR (Number of Unique lazy matched Rules)**
  This metric counts the number of unique lazy matched rules in a module.

- **NCR (Number of Called Rules)**
  This metric counts the number of called rules in a module.

- **DIR (Declarative-Imperative Ratio)**
  This metric calculates the percentage of rules that are hybrid (i.e., include an imperative block).

- **NAR (Number of Abstract Rules)**
  This metric counts the number of abstract rules defined in a module.

- **NHR (Number of Hierarchies of Rules)**
  This metric counts the number of different hierarchies of rules defined in a module.

- **MHH (Maximum Height of a Hierarchy of rules)**
  This metric calculates the height of the highest hierarchy of rules in a module.

- **mHH (minimum Height of a Hierarchy of rules)**
  This metric calculates the height of the lowest hierarchy of rules in a module.

- **AHH (Average Height of Hierarchies of rules)**
  This metric calculates the average height of all hierarchies of rules in a module.

- **MWH (Maximum Width of a Hierarchy of rules)**
  This metric calculates the width of the widest hierarchy of rules in a module.

- **mWH (minimum Width of a Hierarchy of rules)**
  This metric calculates the width of the narrowest hierarchy of rules in a module.

- **AWH (Average Width of a Hierarchy of rules)**
  This metric calculates the average width of all hierarchies of rules in a module.

- **NUH (Number of Unused Helpers)**
  This metric counts the total number of helpers (those defined in a module, but also in any imported library) that are not used.

- **NUL (Number of Unused Lazy matched rules)**
  This metric counts the number of lazy matched rules (unique or not) defined in a module that not used.

- **NUC (Number of Unused Called rules)**
  This metric counts the number of called rules defined in a module that are not used.

- **ASSP (Average Size of Source Patterns)**
  This metric calculates the average number of elements within source patterns across all rules.

- **ASTP (Average Size of Target Patterns)**
  This metric calculates the average number of elements within target patterns across all rules.
- **ABR (Average number of Bindings per Rule)**
  This metric calculates the average number of bindings per rule. The bindings of a rule are all bindings of all target pattern elements of the rule.
- **ANV (Average Number of Variables per rule)**
  This metric calculates the average number of local variables declared per rule.
- **ANF (Average Number of Filtered rules)**
  This metric calculates the average number of rules which have an associated filter condition.
- **NRSE (Number of Rules per Source Element)**
  This metric counts the number of rules that match elements of the same source metaclass.
- **MLC (Maximum Length of nested Calls to helpers)**
  This metric calculates the size of the longest chain of nested calls to helpers, starting from a rule in a module.
- **ALC (Average Length of nested Calls to helpers)**
  This metric calculates the average size of all chains of nested calls to helpers, starting from any rule in a module.
- **NCPC (Number of Clones of a Piece of Code)**
  This metric detects repeated pieces of code and counts the number of times it repeats. Repeated code takes the form of a (set of) binding(s) which is (are) present in different elements of the same out pattern, or more likely, in different out pattern elements of different rules.
- **MLIR (Maximum Length of Implicit dependencies of Rules)**
  Assuming a rule R1 contains a binding whose right side is a (collection of) source element(s), and that (those) element(s) is (are) transformed by rule R2, we say that R1 implicitly depends on R2 because the target element of R1 depends on the target element of R2. In other words, the result of R1 will depend on the result of R2. This metric calculates the longest path in the (directed) graph of implicit dependencies of all rules in a module.
- **NIrT (Number of Invocations to resolveTemp())**
  This metric counts the total number of calls to the resolveTemp() operation.
- **NCrT (Number of Callers to resolveTemp())**
  This metric counts the number of module elements that invoke the resolveTemp() operation.

3.2.2 Library Metrics

The following metrics specifically apply to ATL Libraries:

- **DOL (Dependency of Libraries)**
  This metric counts the number of times a library is imported by other units.
- **FIL (Fan-In of Libraries)**
  This metric counts the number of times a helper defined in a library is used by a helper defined in another library.

- **FOL (Fan-Out of Libraries)**
  This metric counts the number of times a helper defined in a library uses a helper defined in another library.

3.3 Rule Metrics

There is a great number of metrics on rules; however most of them are derived from metrics on modules in the sense that the latter require the former for being computed. This section focuses on metrics that, even if they are derivable, they are interesting per se. Metrics that apply to any variant of Rule are:

- **NLV (Number of Local Variables)**
  This metric counts the number of local variables defined in the rule.
- **NUV (Number of Unused local Variables)**
  This metric counts the number of local variables which are not used in a rule.
- **NUTP (Number of Uninitialized Target Properties)**
  This metric counts the number of properties of all target pattern elements of a rule which are not initialized (neither in a binding nor in a sentence).
- **NCH (Number of Calls to Helpers)**
  This metric counts the number of calls to a helper made by a rule. There are two variants of this metric: calls from bindings and calls made from sentences.
- **NCL (Number of Calls to Lazy matched rules)**
  This metric counts the number of calls to a lazy matched rule made by a rule. There are two variants of this metric: calls from bindings and calls made from sentences.
- **NCC (Number of Calls to Called rules)**
  This metric counts the number of calls to a called rule made by a rule. There are two variants of this metric: calls from bindings and calls made from sentences.
- **NUA (Number of Uses of Attributes)**
  This metric counts the number of times a rule uses an attribute. There are two variants of this metric: uses from bindings and uses made from sentences.

3.3.1 Matched Rule Metrics

The following metrics specifically apply to Matched Rules, and to Lazy Matched Rules as well:

- **NCHF (Number of Calls to Helpers from Filter)**
  This metric counts the number of calls to a helper made by a matched rule from its filter condition.
- **NUAF (Number of Uses of Attributes from Filter)**
  This metric counts the number of times an attribute is used within the filter of a matched rule.

3.3.2 Lazy Matched Rule Metrics

The following metric specifically applies to Lazy Matched Rules:

- **NRC (Number of Received Calls)**
  This metric counts the total number of calls a lazy matched rule receives.

3.3.3 Called Matched Rule Metrics

The following metrics specifically apply to Called Rules:

- **NRC (Number of Received Calls)**
  This metric counts the total number of calls a called rule receives.
- **NOP (Number of Parameters)**
  This metric counts the number of parameters a called rule receives.
- **NUP (Number of Unused Parameters)**
  This metric counts the number of formal parameters declared by a called rule which are not used.

3.4 Helper Metrics

The body of an ATL helper is an ATL Expression, which is essentially an OCL expression. Therefore some metrics for OCL expressions, such as those reported in [6], are applicable to ATL Helpers as well. Metrics that apply to helpers are:

- **NRC (Number of Received Calls)**
  This metric counts the total number of calls a helper receives.
- **NOP (Number of Parameters)**
  This metric counts the number of parameters a helper receives.
- **NUP (Number of Unused Parameters)**
  This metric counts the number of formal parameters declared by a helper which are not used (this includes self).
- **CCH (Cyclomatic Complexity of a Helper)**
  This metric calculates the cyclomatic complexity of a helper.
- **OCH (Order of Complexity of a Helper)**
  This metric calculates the order of (i.e., the big O) the computational complexity of a helper for its worst case, such as linear, quadratic, cubic, etc.
- **NKW (Number of KeyWords)**
  This metric counts the number of ATL keywords occurring in the body of a helper.
- **NOS (Number of Occurrences of Self)**
  This metric counts the number occurrences of self in a helper.
- **NVL (Number of Variables in Let expressions)**
  This metric counts the number of variables defined within let expressions in a helper.
- **NIE (Number of If Expressions)**
  This metric counts the number of if expressions defined in a helper.
- **NBO (Number of Boolean Operators)**
  This metric counts the number of boolean connectives used in a helper.
- **NCO (Number of Comparison Operators)**
  This metric counts the number of comparisons performed in a helper.
- **NIV (Number of Iterator Variables)**
  This metric counts the total number of iterator variables declared in a helper.
- **NAC (Number of Attributes belonging to Context)**
  This metric counts the number of attributes of the context which are used in a helper.

- **NOC (Number of Operations belonging to Context)**
  This metric counts the number of operations of the context that are invoked in a helper.
- **NIO (Number of oclIsTypeOf(), oclIsKindOf() and oclAsType() Operations)**
  This metric counts the number of type cases and downcastings in a helper.
- **NUO (Number of oclIsUndefined() Operation)**
  This metric counts the number of elements which are tested for definedness in a helper.
- **NNR (Number of Navigated References)**
  This metric counts the number of references which are navigated in a helper.
- **NNC (Number of Navigated Classes)**
  This metric counts the number of classes a helper navigates to.
- **WNO (Weighted Number of Operations)**
  This metric counts the number of operations, weighted by their parameters, which occur in a helper. The weight of an operation is the number of its formal parameters.
- **NPM (Number of Parameters typed by a Metamodel element)**
  This metric counts the formal parameters of a helper whose types are elements of any metamodel.
- **MDN (Maximum Depth of Navigations)**
  This metric calculates the length of the longest path of navigations starting from self.
- **NOOC (Number of Operations On Collections)**
  This metric counts the total number of collection operations which are invoked within a helper.

4 Relation of Metrics to Quality Attributes

Metrics are related to quality attributes because different measures for a metric indicate different quality levels with respect to a given attribute. In particular, a metric may be related to one or more quality attribute. For example, a higher value of NIL (number of imported libraries) indicates more reuse, more modularity and also more complexity than a lower one. In that case, we say that NIL is related to Reuse, Modularity and Complexity. In particular, a higher value of NIL has a "positive" effect on such metrics, as it increases the corresponding quality level.

Tables 1 and 2 summarize the effect of each metric specified along Sect. 3 on the quality attributes considered in Sect. 3.1. Symbols '+' and '–' are used for denoting a "positive" or "negative" effect of a higher value of a metric on a quality attribute. In the case of NIL, we have '+' for the three related quality attributes. On the other hand, NHP is only related to Performance, and a higher value negatively affects that quality attribute, and is thus marked with '–', since the application of a helper without parameters is more time consuming than just using attributes. Additionally, the same metric may affect different quality attributes in opposite ways. For example, a higher value of ASSM (average size of source metamodels) would make a transformation definition less understandable and less modifiable, but more complex.

As expressed before, the same metric may affect more than one quality attribute, while a quality attribute may also be affected by more than one metric. Therefore, it is interesting to see how many metrics affect each quality attribute. Alternatively, we may calculate the percentage of metrics from the proposed set affecting each quality attribute.

Id	Metric	Quality Attributes									
		Understandability	Modifiability	Reusability	Reuse	Modularity	Completeness	Consistency	Conciseness	Complexity	Performance
Unit Metrics											
NIL	Number of Imported Libraries				+	+				+	
TIL	Total number of Imported Libraries				+	+				+	
NH	Number of Helpers	–	–	+						+	
NHP	Number of Helpers without Parameters										–
BOU	Balance Of a Unit	+		+		+					
NHC	Number of Helpers per Context									+	
NOH	Number of Overloaded Helpers	+									
Module Metrics											
NSM	Number of Source Models		–	–						+	
NTM	Number of Target Models		–	–							
ASSM	Average Size of Source Metamodels	–	–							+	
ASTM	Average Size of Target Metamodels	–	–							+	
CSM	Coverage of Source Metamodels						+			+	–
CTM	Coverage of Target Metamodels						+			+	–
NA	Number of Attributes	–	–								
TA	Transformation Approach (value: declarative)	+	+							–	
NMR	Number of Matched Rules	–	–							+	
NLR	Number of Lazy matched Rules	–	–							+	
NUR	Number of Unique lazy matched Rules	–	–							+	
NCR	Number of Called Rules	–	–							+	
DIR	Declarative/Imperative Ratio	–	–							+	
NAR	Number of Abstract Rules	–	+		+				+	+	
NHR	Number of Hierarchies of Rules	–	+		+				+	+	
MHH	Maximum Height of a Hierarchy of rules				+				+	+	
mHH	minimum Height of a Hierarchy of rules										
AHH	Average Height of Hierarchies of rules				+				+	+	
MWH	Maximum Width of Hierarchies of rules				+				+	+	
mWH	minimum Width of Hierarchies of rules										
AWH	Average Width of Hierarchies of rules				+				+	+	
NUH	Number of Unused Helpers	–								+	–
NUL	Number of Unused Lazy matched rules	–								+	–
NUC	Number of Unused Called rules	–								+	–
ASSP	Average Size of Source Patterns	–	–					–		+	
ASTP	Average Size of Target Patterns	–	–					–		+	
ABR	Average number of Bindings per Rule	–	–				+	–		+	
ANV	Average Number of Variables per rule	+	+					–		+	
ANF	Average Number of Filtered rules	–	–							+	
NRSE	Number of Rules per Source Element	–	–							+	
MLC	Maximum Length of nested Calls to helpers	–	–							+	
ALC	Average Length of nested Calls to helpers	–	–							+	
NCPC	Number of Clones of a Piece of Code		–		–			–	–	+	
MLIR	Maximum Length of Implicit dependencies of Rules	–	–							+	
NIrT	Number of Invocations to resolveTemp()	–								+	
NCrT	Number of Callers to resolveTemp()	–								+	

Table 1: Relation of metrics to quality attributes (part 1)

This could be used for motivating the search for more metrics that affect a certain quality attribute with a low ratio, or even discarding metrics which address a quality attribute with a high ratio and, for example, could be hard to implement or interpret. Table 3 shows, for each quality attribute, the percentage of the metrics that affect it, either positively or negatively.

Id	Metric	Quality Attributes									
		Understandability	Modifiability	Reusability	Reuse	Modularity	Completeness	Consistency	Conciseness	Complexity	Performance
Library metrics											
DOL	Dependency of Libraries				+						
FIL	Fan-In of Libraries			+							
FOL	Fan-Out of Libraries				+						
Rule metrics											
NLV	Number of Local Variables	–	–							+	
NUV	Number of Unused local Variables							–	–		
NUTP	Number of Uninitialized Target Properties						–		–		
NCH	Number of Calls to Helpers	+			+			–		+	–
NCL	Number of Calls to Lazy matched rules	–								+	
NCC	Number of Calls to Called rules	–								+	
NUA	Number of Uses of Attributes	+			+					+	+
Matched Rule metrics											
NCHF	Number of Calls to Helpers from Filters	+			+						–
NUAF	Number of Uses of Attributes from Filters	+			+						
Lazy Matched Rule metrics											
NRC	Number of Received Calls				+					+	
Called Rule metrics											
NRC	Number of Received Calls				+					+	
NOP	Number Of Parameters	–		–							
NUP	Number Of Unused Parameters	–							–		
Helper metrics											
NRC	Number of Received Calls				+						
NOP	Number Of Parameters	–		–							
NUP	Number of Unused Parameters	–							–		
CCH	Cyclomatic Complexity of a Helper	–								+	
OCH	Order of Complexity of a Helper	–								+	+
NKW	Number of KeyWords	–	–							+	
NOS	Number of Occurrences of Self	–	–							+	
NVL	Number of Variables in Let expressions	–	–							+	
NIE	Number of If Expressions	–	–							+	
NBO	Number of Boolean Operators	–	–							+	
NCO	Number of Comparison Operators	–	–							+	
NIV	Number of Iterator Variables	–	–							+	+
NAC	Number of Attributes belonging to Context	–	–							+	
NOC	Number of Operations belonging to Context	–	–							+	
NIO	Number of calls to oclIsTypeOf, etc.	–	–					–		+	
NUO	Number of calls to oclIsUndefined	–	–					+		+	
NNR	Number of Navigated References	–	–							+	
NNC	Number of Navigated Classes	–	–							+	
WNO	Weighted Number of Operations	–	–							+	
NPM	Number of Parameters typed by Model element	–	–	+						+	
MDN	Maximum Depth of Navigations	–	–		–					+	
NOOC	Number of Operations on Collections	–	–		+					+	+

Table 2: Relation of metrics to quality attributes (part 2)

5 Conclusions

In this paper we presented a set of metrics which may be used for measuring model transformations defined in ATL. Measures obtained from the application of these metrics then enable assessing the quality of ATL transformations. We consider quality in terms of a set of quality attributes identified elsewhere [5, 8] as relevant for the context of model transformations. Metrics apply to both ATL modules and ATL libraries, however

Quality Attribute	Relation to metrics
Understandability	72%
Modifiability	52%
Reusability	10%
Reuse	25%
Modularity	4%
Completeness	5%
Consistency	11%
Conciseness	14%
Complexity	77%
Performance	15%

Table 3: Distribution of metrics

we defined metrics which are to be applied to the complete definition of a transformation (i.e., a module and the transitive closure of all imported libraries).

Metrics were derived from the ATL metamodel based on the intuition of what properties of an ATL transformation could be interesting to measure. The set of metrics we proposed is not intended to be exhaustive, and furthermore, it needs to be tuned in many dimensions. To begin with, the practical applicability of the metrics requires the ability to properly interpret the measures obtained from them. That is, the ability to understand the meaning of specific ranges of values. This could be achieved by a theoretical analysis of the metrics, complemented with a series of experiments. As an additional result of this, the actual contribution of individual metrics could be better understood. This in turn may lead to the definition of composite higher-level metrics from existing ones, which may be more naturally interpreted. In particular, our set (81 metrics) is larger than that for ASF+SDF in [8] (27 metrics). This could be explained by the size and complexity of the ATL metamodel. It could also indicate that our set has too many fine-grained metrics, even though the abstraction level of both sets is similar. Values from Table 3 suggest that additional metrics addressing Modularity and Completeness would be required. Additionally, since ATL transformations are unidirectional, they can be regarded as functions on models, and therefore our set of metrics may benefit from porting the notion of function point to the model transformation context.

Either for their application in real world contexts or for conducting experiments for enhancing our proposal, metrics need to be implemented. One possible approach to that is by means of a model transformation, analogous to that used in [9] for measuring model repositories, that generates a Measure model from a (set of) ATL unit(s). Such a transformation would match different elements within the definition of an ATL transformation for producing measure elements holding the resulting value of computing metrics. In this work we informally specified the meaning of each metric in natural language. Such specification could have been formulated as OCL expressions associated to the ATL metamodel. Each of those expressions would then constitute the body of a helper which computes the corresponding metric.

References

[1] ATL Transformations Zoo. Internet: `http://www.eclipse.org/m2m/atl/atlTransformations/`, 2009.

[2] Atlantic Zoo. Internet: `http://www.eclipse.org/gmt/am3/zoos/atlanticZoo/`, 2009.

[3] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A Model Transformation Tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.

[4] F. Jouault and I. Kurtev. Transforming Models with ATL. In J.-M. Bruel, editor, *MoDELS Satellite Events*, volume 3844 of *Lecture Notes in Computer Science*, pages 128–138. Springer, 2005.

[5] P. Mohagheghi and V. Dehlen. Developing a Quality Framework for Model-Driven Engineering. In H. Giese, editor, *MoDELS Workshops*, volume 5002 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2007.

[6] L. Reynoso, M. Genero, and M. Piattini. Towards a Metric Suite for OCL Expressions Expressed within UML/OCL Models. *Journal of Comptuer Science and Technology*, 4(1):38–44, 2004.

[7] M. Saeki and H. Kaiya. Measuring Model Transformation in Model Driven Development. In J. Eder, S. L. Tomassen, A. L. Opdahl, and G. Sindre, editors, *CAiSE Forum*, volume 247 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[8] M. F. van Amstel, C. F. J. Lange, and M. G. J. van den Brand. Metrics for Analyzing the Quality of Model Transformations. In G. Falcone, Y.-G. Guéhéneuc, C. F. J. Lange, Z. Porkoláb, and H. A. Sahraoui, editors, *12th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2008)*, pages 41–51, Paphos, Cyprus, July 2008.

[9] E. Vépa, J. Bézivin, H. Brunelière, and F. Jouault. Measuring Model Repositories. In *2nd Workshop on Model Size Metrics, co-located with MoDELS 2006 (MSM 2006)*, Genova, Italy, October 2006.

[10] A. Vignaga. Paraphrasing Reference Models and Transformations. Technical Report TR/DCC-2009-3, Computer Science Department, Universidad de Chile, 2009.