A Customizable Approach for the Automated Quality Assessment of Modelling Artefacts

Francesco Basciani*, Juri Di Rocco*, Davide Di Ruscio*, Ludovico Iovino† and Alfonso Pierantonio*
* DISIM - University of L’Aquila
Via Vetoio snc, I–67100, L’Aquila, Italy
Email: name.surname@univaq.it
† Gran Sasso Science Institute
Via F. Crispi 7, I–67100, L’Aquila, Italy
Email: name.surname@gssi.infn.it

***Abstract*—In Model-Driven Engineering (MDE) giving a precise definition of quality models, identifying which quality attributes are of interest for specific stakeholders, and how relating and aggregating together quality attributes are still open issues. The main limitations of currently available quality approaches are *limited extensibility*, *artifact specificity*, and *manual assessment*. This paper proposes an approach supporting the definition of custom quality models consisting of hierarchically organized quality attributes whose evaluation depends on metrics specifically conceived and applied on the modeling artifacts to be analysed. A domain specific language is proposed to specify how quality attributes and metrics have to be aggregated. An execution environment is also provided to apply the defined quality models on actual modeling artifacts so to enable their automated quality assessment. Real applications of the approach are presented by defining and applying explanatory quality models suitably conceived to assess the quality of metamodels and transformations retrieved from public repositories.**

I. Introduction

Software Quality Engineering [14] is a discipline that is concerned with improving the approach to software quality. What exactly constitutes the quality of software is often subject to debate due to the several perspectives throughout the software lifecycle. In this context, placing reliance on software quality models is well-accepted in order to support quality management of software systems [20]. A common approach to formulate a software quality model is to first identify a small set of high-level quality attributes and then decompose them into sets of subordinate attributes. Over the years, researchers have proposed new models (e.g., [4, 7, 11, 12, 18]) to emphasize the need for an appropriate support to perform quality checks while developing software systems. Despite successes and standardisation efforts, none of the proposed models seems to have reached a significant acceptance. To some extent, this is due to the different meanings they can be given to depending on the considered application scenarios and intended purpose [5, 11].

The need for approaches and tools supporting quality assessment also in Model-Driven Engineering [22] (MDE) is witnessed by the increasing interest around the topic and by a large corpus of research that has been produced over the last few years (e.g., see [3, 16, 17]). MDE offers a promising approach to alleviate the software complexity by shifting the focus from coding to modeling. Most of the existing approaches supporting quality assessment in MDE identify a number of artefact-specific metrics (e.g., for metamodels, class diagrams, and model transformations) and discuss how they characterize the analysed artefacts in terms of modularity, completeness, and correctness. Even though some of the approaches are tool supported, few of them permit to automatically assess quality, and the provided quality models cannot be extended or customized to better accommodate specific requirements. In particular, as discussed later in the paper, the limitations affecting currently available techniques are related to their *limited extensibility*, *artifact specificity*, and *manual assessment*.

This research was supported by the EU through the Model-Based Social Learning for Public Administrations (Learn Pad) FP7 project (619583)

In this paper, we present an extensible and generic approach for the quality assessment of modeling artifacts. The approach consists of a tool chain supporting the specification of custom quality models (which best fit particular business contexts) and the automated quality assessment of any kind of modeling artifact. The specification of quality models and the way quality attributes defined therein have to be hierarchically aggregated and thus evaluated is performed by means of a proposed domain specific language (making use of OCL[1]).

Outline. The paper is structured as follows. Section II introduces the problem of assessing the quality in model-driven engineering and gives an overview of the related work. The proposed quality assessment approach is presented Section III. Concrete applications of it on real metamodels and transformations are given in Section IV. Section V concludes the paper and outlines some future plans.

II. Quality Assessment in Model-Driven Engineering

Over the last decade, several approaches have been proposed to support the quality measurement of modeling artifacts. In [3] authors introduce a quality model specifically conceived to measure the quality of metamodels: characteristics like *maintainability*, *portability*, and *usability* are introduced together with sub-characteristics like *analyzability*, *adaptability*, and *understandability* for each main characteristic.

The *mmSpec* language is proposed in [16] to specify and check metamodel properties. A library of 30 properties has been proposed by organizing them in four categories namely *design*, *best practices*, *naming conventions*, and *metrics*. Example of metamodel characteristics that can be expressed by means of *mmSpec* are:

1) there are no composition cycles (from the design category);
2) there are no uninstantiable classes, i.e., abstract classes without children (from the best practices category);
3) element names are too complex to process, i.e., too long (from the naming conventions category);
4) no class is overloaded with attributes, 10-max by default (from the metrics category).

In [17] quality model specifically conceived for metamodels is proposed. Starting from the formulation in [3], each quality attribute is aligned with a combination of metrics supposed to measure

[1]OMG Object Constraint Language: http://www.omg.org/spec/OCL/

it. Examples of considered quality attributes are *reusability*, *understability*, *functionality*, and *extendibility*. Their measurement is based on proposed formulas involving typical object-oriented measures like *coupling*, *size*, and *inheritance*.

An empirical study has been conducted in [10] with the aim of understanding the perception of quality that modelers have about specific metamodels. The study has involved 10 metamodel characteristics like *modularity*, *completeness*, *understandability*, *consistency*, and *correctness*. Correlations among them have been identified suggesting that the perceived quality *"was mainly driven by the metamodel completeness, correctness and modularity while other quality attributes could be neglected"* [10].

Quality characteristics of model transformations have been also investigated. In particular, in [21] a systematic evaluation framework for comparing model transformation approaches is provided. The framework relies on the quality characteristics of the ISO/IEC 9126-1 standard [12]. For each quality attribute an alignment with metrics and evaluation criteria is introduced. In [23] a catalogue of design patterns for developing model transformations is given. The proposed catalogue has been evaluated with respect to quality characteristics like *correctness*, *efficiency*, *reliability*, and *maintainability*. Such properties are formally encoded to support their automated evaluation by means of model checking techniques. Other works focus on metrics specifically conceived to measure structural characteristics of metamodels, models, and model transformations (see [15], [1], [26], respectively). In [15] a number of metrics are introduced to calculate the ratio between the number of metaclasses in a given metamodel and the number of concepts that are explicitly available in the concrete syntax of the corresponding modeling language. EMF Metrics [1] is a tool supporting the specification and calculation of metrics for models developed atop of the Eclipse Modeling Platform (EMF). Number of attributes, classes, and associations are examples of metrics defined by means of EMF Metrics and discussed in [1]. A set of metrics for measuring ATL [13] model transformations is given in [1] without giving any categorization of them. Number of matched rules, helpers, and bindings are examples of metrics discussed in [26].

Despite the large corpus of research outlined above testifies the increasing interest on the topic of quality in MDE, existing approaches suffer all or some of the following shortcomings:

▷ *limited extensibility:* depending on the application domain and user needs, the required quality characteristics supporting the evaluation of a given modeling artifact might change. Consequently, a quality assessment approach should be open to custom quality models that users can easily define to support the quality assessment problem at hand. For instance, the approach presented in [16] goes in that direction, even though *mmSpec* permits to specify only structural metamodel characteristics without providing the means to hierarchically organize them as required when defining quality models;

▷ *artifact specificity:* all the approaches previously presented have been defined to support the quality assessment of specific artifacts i.e., metamodels, models, or transformations. Thus, users that would like to measure all the modeling artifacts involved in a given model-based process should adopt different approaches and tools. For instance, modelers that have developed metamodels and transformations and would like to assess their quality should adopt at least two different approaches like [17, 21];

▷ *manual assessment:* some of the work outlined above introduces quality characteristics without providing the tools for automatically measuring the introduced quality attributes as e.g., in the case of [3].

In our opinion, a quality assessment approach should overcome all the limitations previously discussed as presented in the next section.

III. The proposed quality assessment approach

To overcome the limitations previously discussed, the quality assessment approach shown in Fig. 1 is proposed. It relies on a *quality metamodel* enabling the specification of quality models consisting of hierarchically organized quality characteristics. Each quality attribute consists of an expression defining how the values of the sub-attributes have to be aggregated. The leafs of the quality model consist of metrics whose values are calculated by dedicated *metrics providers*. By resembling concepts from the "models at run-time" research area [19], the *evaluation engine* is able to apply quality models on source artifacts by executing all the aggregating expressions defined for each quality characteristic. To perform the specified measurements the approach has been implemented atop of the MDEForge platform [2] providing the means to store and retrieve modeling artifacts further than mechanisms to specify and execute metrics [6].

It is important to remark that the proposed approach is mainly intended for modelers and developers that have the availability of reusable modeling artifacts and that want to make an evidence-based decision on which one should be selected for the task at hand. Moreover, the approach is artifact independent in the sense that it can be applied to assess the quality of any kind of modeling artifacts. We are not aiming at proposing ”yet another quality model”, but an holistic approach providing users with the means to define and customize their own quality idea. Depending on the application domain and needs, users will be able to define its own quality model and even extend the set of available metric providers that are required to measure specific quality attributes, which are considered relevant by users.

The main building blocks of the approach shown in Fig. 2 (i.e., the *quality metamodel* and the *evaluation engine*) are described in the next sections.

A. The Quality Metamodel

It plays a key role in the proposed approach since it enables the specification of quality models according to user requirements. The metamodel is shown in Fig. 2 and consists of a number of constructs as explained in the following. `QualityModel` is the root element consisting of `QualityAttributes`, `ValueTypes`, and `MetricProviders`.

A `QualityAttribute` represents a quality aspect that is considered to be relevant for contributing to the quality assessment of a given artifact. A quality attribute, like *maintainability* can be an aggregation of other attributes, like *changeability* and *modularity*. Thus, each quality attribute specifies how to aggregate the contained

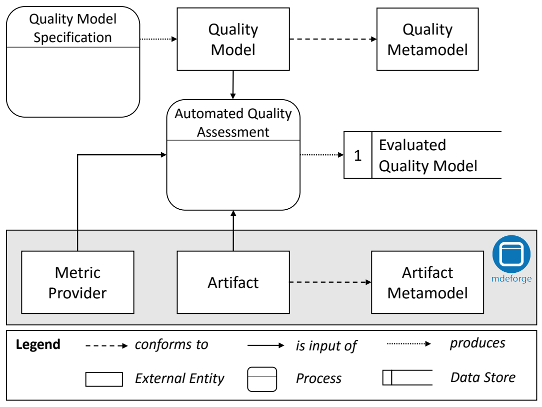


Fig. 1. The proposed quality assessment approach

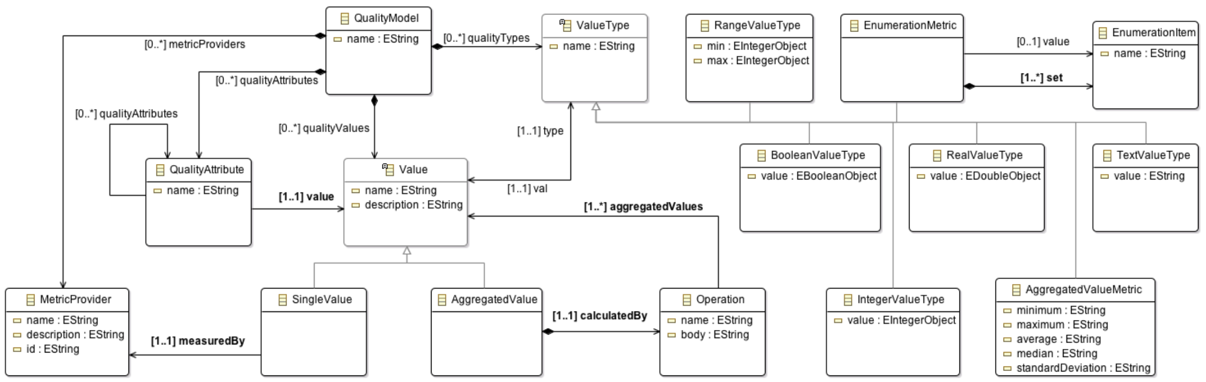


Fig. 2. The Quality Metamodel

attributes in order to provide an overall quality value for the considered attribute as described below.

Each quality attribute has a `Value` representing its calculated value. `Value` is abstract and it can be `SingleValue` or `AggregatedValue`. `SingleValue` represents the value obtained by the application of a given `MetricProvider`, which refers to the software component able to calculate a specific metrics. The attribute `id` of a given `MetricProvider` is used to identify and retrieve the actual software component implementing the considered metric. `AggregatedValue` indicates a composition of different values. An aggregation is specified by means of an OCL expression contained in the attribute `body` of the `Operation` element.

Each `Value` has a reference with a `ValueType` element, which defines its type. `ValueType` is abstract and several specializations are provided in order to enable the specifications of categories (like *low*, *medium*, *high*), ranged values (e.g., from 0 to 5), textual, boolean, integer, and real values. Additionally, `AggregatedValueMetric` permits to specify aggregated values for a given metric like average, median, and standard deviation.

B. The Evaluation Engine

Given a source artifact to be measured (e.g., a model transformation or a metamodel) and an input quality model, the evaluation engine executes all the metrics and expressions therein by producing at the end of the process the *evaluated quality model*. A fragment of the Java implementation of the evaluation engine is shown in Listing 1. It relies on the Eclipse Modeling Framework (EMF) and the method `actualize` at line 3 performs the actualization of a previously retrieved quality model on a given artifact. For each quality attribute the corresponding value is calculated. In case of single values the corresponding metric providers are executed and the calculated values are stored (see lines 6-15). In case of aggregated values (see lines 17-21), the corresponding OCL expressions are evaluated by means of the method `evaluateOCL` shown at lines 26-32.

The method `compute` in the sample `Main` class at line 37 retrieves the quality model `qm`, and actualizes it on the model transformation `UML22Measure.atl`. The evaluated quality model is serialized in the file `evaluatedQM.xmi`.

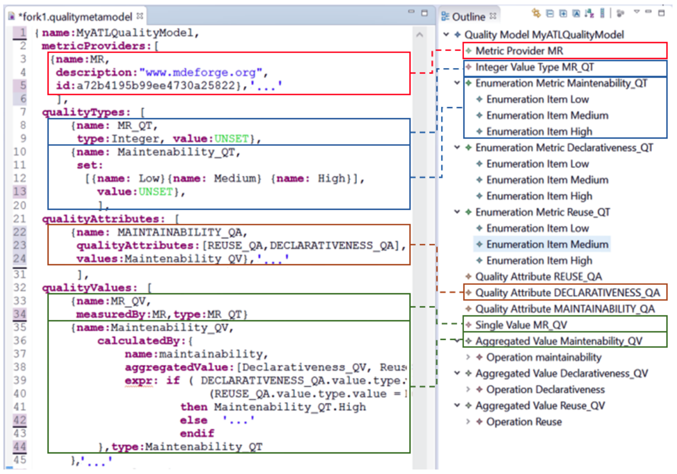
Listing 1. Fragment of the Evaluation Engine

```
 1 class EvaluationEngine {
 2  ...
 3  private QualityModel actualize(QualityModel qm, String
        artifactName) throws Exception {
 4   ...
 5   for (Value value : qm.getQualityValues()) {
 6    if(value instanceof SingleValue) {
 7     SingleValue sv = (SingleValue) value;
 8     MetricProvider mp = sv.getMeasuredBy();
 9     Metric metric = artifactService.
          getArtifactMetricByArtifactAndMetricNames(
          artifactName, mp.name));
10     if(sv.getTypes() instanceof IntegerValueType) {
11     IntegerValueType vt = (IntegerValueType)sv.getTypes();
12     vt.setValue(((IntegerMetric)metric).getValue());
13     }
14     ...
15    }
16    ...
17    if(value instanceof AggregatedValue) {
18     AggregatedValue av = (AggregatedValue) value;
19     OCLExpression body = av.getCalculatedBy().getBody();
20     av.setTypes(evaluateOCL(qm,body));
21    }
22   }
23   ...
24   return qm;
25  }
26  public static Value evaluateOCL(QualityModel qm,
        OCLExpression oclExpr) throws ParserException {
27   OCL ocl = OCL.newInstance(EcoreEnvironmentFactory.
         INSTANCE);
28   ...
29   Query<EClassifier, EClass, EObject> query = ocl.
         createQuery(oclExpr);
30   Value result = (Value) query.evaluate(qm);
31   return result;
32  }
33 }
34 ...
35 class Main {
36 ...
37  public void compute(String qmPath){
38   QualityModel qm = getQualityModel(qmPath);
39   QualityModel evaluatedQM = actualize(qm, "UML22Measure.
         atl");
40   serialize(evaluatedQM, "evaluatedQM.xmi");
41  }
42 }
```

Next section presents real applications of the proposed approach to assess the quality of actual model transformations and metamodels stored in the MDEForge repository [2].

IV. Automated Quality Assessment of Metamodels and Model Transformations

In this section we present the application of the proposed approach to assess the quality of metamodels and transformations by considering selected quality characteristics among those presented in [24] and [26], respectively. The selection has been done to enable the specification of explanatory quality attributes permitting to highlight the strengths of the approach. By using the metamodel shown in Fig. 2, we have defined a quality model to measure the quality of model transformations and we have applied it on ≈110 transformations stored in the MDEForge repository (see Section IV-A). Another quality model has been defined to measure the quality of metamodels



```
1  { name:MyATLQualityModel,
2   metricProviders:[
3    {name:MR,
4     description:"www.mdeforge.org",
5     id:a72b4195b99ee4730a25822},'...'
6    ],
7   qualityTypes: [
8       {name: MR_QT,
9        type:Integer, value:UNSET},
10      {name: Maintenability_QT,
11       set:
12        [{name: Low}{name: Medium} {name: High}],
13         value:UNSET},
20         ],
21  qualityAttributes: [
22      {name: MAINTAINABILITY_QA,
23       qualityAttributes:[REUSE_QA,DECLARATIVENESS_QA],
24       values:Maintenability_QV},'...'
31       ],
32  qualityValues: [
33      {name:MR_QV,
34       measuredBy:MR,type:MR_QT}
35      {name:Maintenability_QV,
36          calculatedBy:{
37              name:maintainability,
38              aggregatedValue:[Declarativeness_QV, Reus
39              expr: if ( DECLARATIVENESS_QA.value.type.
40                       (REUSE_QA.value.type.value = 
41                  then Maintenability_QT.High
42                  else  '...'
43                  endif
44          },type:Maintenability_QT
45      },'...'
```

```
Quality Model MyATLQualityModel
    Metric Provider MR
    Integer Value Type MR_QT
    Enumeration Metric Maintenability_QT
        Enumeration Item Low
        Enumeration Item Medium
        Enumeration Item High
    Enumeration Metric Declarativeness_QT
        Enumeration Item Low
        Enumeration Item Medium
        Enumeration Item High
    Enumeration Metric Reuse_QT
        Enumeration Item Low
        Enumeration Item Medium
        Enumeration Item High
    Quality Attribute REUSE_QA
    Quality Attribute DECLARATIVENESS_QA
    Quality Attribute MAINTAINABILITY_QA
    Single Value MR_QV
    Aggregated Value Maintenability_QV
        Operation maintainability
    Aggregated Value Declarativeness_QV
        Operation Declarativeness
    Aggregated Value Reuse_QV
        Operation Reuse
```

Fig. 3. Sample Quality Model for ATL Model Transformations

and we have applied it on ≈500 metamodels stored in the same repository (see Section IV-B). The obtained data are discussed in the following and interested readers can refer to the complete data available online[2].

A. Quality Assessment of Model Transformations

In [26] the author defines a set of metrics specifically defined for measuring structural characteristics of ATL model transformations [13]. Examples of metrics discussed in [26] are the following:

- *TR*: number of transformation rules;
- *MR*: number of matched rules (excluding lazy matched rules);
- *Bind*: number of bindings;
- *RT*: number of *resolveTemp()* calls;
- *ATR*: number of abstract transformation rules;
- *HR*: number of rule hierarchies;
- HHR_{Max}: maximum height of rule hierarchies;
- WHR_{Max}: maximum width of rule hierarchies.

ATL is a rule based language with mixture of declarative and imperative constructs. An interesting characteristic that can be considered when assessing the quality of ATL transformations is their *declarativeness*, which can contribute to assess their *understability*. By considering the metrics above and by borrowing the definition of declarativeness given in [25] the following quality attribute can be defined:

$$Declarativeness = \begin{cases} \text{Low} & \text{if } \frac{MR}{TR} + \frac{RT}{Bind} < 0.3 \\ \text{Medium} & \text{if } 0.3 =< \frac{MR}{TR} + \frac{RT}{Bind} < 0.7 \\ \text{High} & \text{if } \frac{MR}{TR} + \frac{RT}{Bind} >= 0.7 \end{cases} \quad (1)$$

According to such a definition, ATL model transformations can be of low, medium, and high declarativeness by referring to imperative, hybrid, and declarative transformations, respectively. To assess the declarativeness of ATL model transformations the metrics *MR*, *RT*, *TR*, and *Bind* have been used and the thresholds 0.3 and 0.7 have been considered. It is important to remark that such thresholds are only explanatory and we are not arguing about their correctness, which should be empirically validated. Such a validation activity is beyond the scope of this paper. By relying on the same metrics, and by borrowing concepts from [26], the *Reuse* attribute can be also defined as follows with the aim of characterizing to what extent a given ATL transformation reuses already existing rules.

$$reuseValue = (\frac{ATR + HR + HHR_{Max} + WHR_{Max}}{TR}) \times \frac{1}{4}$$

$$Reusability = \begin{cases} \text{Low} & \text{if } \mathit{reuseValue} < 0.2 \\ \text{Medium} & \text{if } 0.2 =< \mathit{reuseVale} < 0.3 \\ \text{High} & \text{if } \mathit{reuseValue} >= 0.3 \end{cases} \quad (2)$$

In [9], authors define maintainability as "*the ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment*". Accordingly, the attributes *Reuse* and *Declarativeness* can be combined to define the *Maintainability* quality of model transformations as follows:

$$Maintainability = \begin{cases} \text{High} & \text{if } (\mathit{Declarativeness} = \mathit{High} \\ & \text{and } \mathit{Reuse} = \mathit{High}) \text{ or} \\ & (\mathit{Declarativeness} = \mathit{High} \\ & \text{and } \mathit{Reuse} = \mathit{Medium}) \\ \text{Medium} & \text{if } (\mathit{Declarativeness} = \mathit{High} \\ & \text{and } \mathit{Reuse} = \mathit{Low}) \text{ or} \\ & (\mathit{Declarativeness} = \mathit{Hybrid} \\ & \text{and } \mathit{Reuse} = \mathit{High}) \text{ or} \\ & (\mathit{Declarativeness} = \mathit{Hybrid} \\ & \text{and } \mathit{Reuse} = \mathit{Medium}) \\ \text{Low} & \text{otherwise} \end{cases} \quad (3)$$

The quality attributes previously presented can be defined by means of the textual domain specific language, which has been conceived to support the specification of quality models conforming to the metamodel shown in Fig. 2. In particular, Fig. 3 shows a sample quality model defined by means of the conceived textual language and containing the specifications of the quality attributes *Declarativeness*, *Reuse* and *Maintainability* previously presented. Because of lack of space only a fragment of the whole quality model is shown. The right-hand side of Fig. 3 shows the outline of the quality model textually defined on the left-hand side of the same figure.

The model starts with the definition of the quality model element *MyATLQualityModel* and the declaration of the used metric providers like the one at lines 2-6 in Fig. 3. After the metric provider declarations, the model contains the specification of all the value types used in the model, like the enumeration specifically introduced at lines 10-13 to define different kinds of maintainability of an ATL model transformation, which can be low, medium or high. From lines 21 to 31 the actual definition of the maintainability quality attribute is defined as the aggregated value of the other declared quality attributes *Reuse* and *Declarativeness*. The definition of the expression to be evaluated for measuring the value of the maintainability attribute is specified at lines 36-43. In particular, the keyword *calculatedBy* defines the OCL expression as shown at lines 39-43 in Listing 2.

Listing 2. OCL expressions specifying the aggregation of quality attributes defining the *Maintainability* quality attribute

```
1 let  Maintainability =
2 if ( DECLARATIVENESS_QA.value.type.value =
      Declarativeness_QT.High and
3    (REUSE_QA.value.type.value = Reuse_QT.High))
4  then Maintenability_QT.High
5  else  if (DECLARATIVENESS_QA.value.type.value =
       Declarativeness_QT.High and
6          REUSE_QA.value.type.value = Reuse_QT.Medium)
7     then Maintainability_QT.High
8     else if(DECLARATIVENESS_QA.value.type.value =
          Declarativeness_QT.High and
9          REUSE_QA.value.type.value = Reuse_QT.Low)
10     then Maintainability_QT.Medium
```

[2]http://bit.ly/1NsbEds

```
        else if (DECLARATIVENESS_QA.value.type.value =
              Declarativeness_QT.Medium and
              REUSE_QA.value.type.value = Reuse_QT.High)
         then Maintenability_QT.Medium
         else if (DECLARATIVENESS_QA.value.type.value =
               Declarativeness_QT.Medium and
               REUSE_QA.value.type.value = Reuse_QT.Medium)
          then Maintenability_QT.Medium
          else Maintenability_QT. Low
         endif
        endif
       endif
      endif
endif
...
let Reuse =
if (((ATR_QV.type.value + HHR_Max_QV.type.value +
        WHR_Max_QV.type.value)/4) < 0.2)
then Reuse_QT.Low
else if (((ATR_QV.type.value + HHR_Max_QV.type.value +
         WHR_Max_QV.type.value)/4)<0.3)
     then Reuse_QT.Medium
     else Reuse_QT.High
    endif
endif
...
let declarativeness =
if ((MR_QV.type.value/TR_QV.type.value +
        RT_QV.type.value + Bind_QV.type.value)<0.3)
    then Declarativeness_QT.Imperative
    else if ((MR_QV.type.value / TR_QV.type.value +
        RT_QV.type.value + Bind_QV.type.value)<0.7)
     then Declarativeness_QT.Hybrid
     else Declarativeness_QT.Declarative
    endif
endif
```

TABLE I
SAMPLE OF THE ANALYSED MODEL TRANSFORMATIONS

ATL Transformation	Declarativeness	Reuse	Maintainability
Uml22Measure	Low	Low	Low
Table2TabularHTML	Low	Medium	Low
KM32Measure	Low	Low	Low
MMD2ATL	Medium	Low	Low
WSDL2R2ML	Medium	Medium	Medium
Petrinet_2_PNML	Medium	Medium	Medium
StateMachine2calculusSystem	High	Medium	High
HTML2XML	High	Medium	High
Families2Persons	High	High	High
AnyLogic2XML	High	Medium	High
MM0Transf	High	High	High
TextualPathExp2PathExp	High	Medium	High

The quality model shown in Fig. 3 has been applied on a corpus consisting of ≈100 ATL transformations retrieved from the MDE-Forge repository, and a sample of the obtained results is shown in Table I. According to table, *TextualPathExp2PathExp*[3] is the most maintainable transformation in the analysed corpus. On the contrary, *UML22Measure*[4] resulted to be the less maintainable. In fact, it consists of 9 rules in total, and only 1 is a matched one. Moreover, according to the collected data the transformation has a low reuse degree and for this has been classified as difficult to maintain.

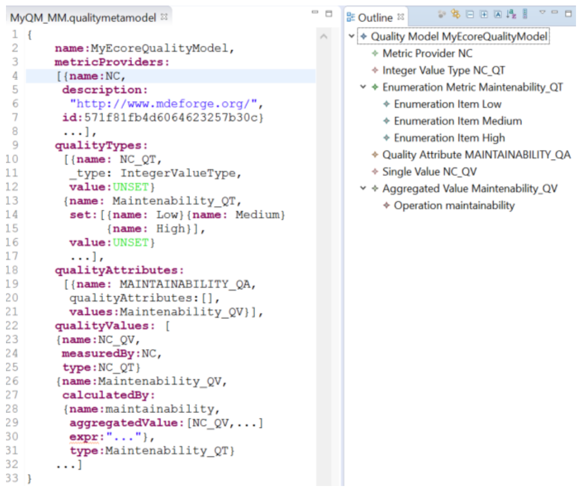
B. Quality Assessment of Metamodels

In this section we show the application of the proposed approach to assess the quality of metamodels. To this end we borrowed the definition of the *maintainability* attribute given in [8] and defined for class diagrams in terms of the following metrics:

- *NC*: number of classes;

[3]https://www.eclipse.org/atl/atlTransformations/PathExp2PetriNet/ExamplePathExp2PetriNet[v00.01].pdf

[4]http://www.eclipse.org/atl/atlTransformations/UML22Measure/ExampleUML22Measure[v00.02].pdf


MyQM_MM.qualitymetamodel

```
1  {
2      name:MyEcoreQualityModel,
3      metricProviders:
4      [{name:NC,
5       description:
6        "http://www.mdeforge.org/",
7       id:571f81fb4d6064623257b30c}
8       ...],
9      qualityTypes:
10      [{name: NC_QT,
11        _type: IntegerValueType,
12        value:UNSET}
13      {name: Maintenability_QT,
14        set:[{name: Low}{name: Medium}
15             {name: High}],
16        value:UNSET}
17        ...],
18     qualityAttributes:
19      [{name: MAINTAINABILITY_QA,
20        qualityAttributes:[],
21        values:Maintenability_QV}],
22     qualityValues: [
23     {name:NC_QV,
24      measuredBy:NC,
25      type:NC_QT}
26     {name:Maintenability_QV,
27      calculatedBy:
28       {name:maintainability,
29        aggregatedValue:[NC_QV,...]
30        expr:"..."},
31        type:Maintenability_QT}
32     ...]
33 }
```

Outline
- Quality Model MyEcoreQualityModel
  - Metric Provider NC
  - Integer Value Type NC_QT
  - Enumeration Metric Maintenability_QT
    - Enumeration Item Low
    - Enumeration Item Medium
    - Enumeration Item High
  - Quality Attribute MAINTAINABILITY_QA
  - Single Value NC_QV
  - Aggregated Value Maintenability_QV
    - Operation maintainability

Fig. 4. Sample Quality Model for Metamodels

- *NA*: number of attributes;
- *NR*: number of references;
- DIT_{Max}: it is the maximum between the *DIT* value obtained for each class of the metamodel. The *DIT* value for a class within a generalization hierarchy is the longest path from the class to the root of the hierarchy;
- $HAgg_{Max}$: it is the maximum between the *HAgg* value obtained for each class of the metamodel. The *HAgg* value for a class within an relation chain is the longest path from the class to others.

By relying on such metrics, the maintainability quality attribute can be defined as follows:

$$
\begin{gathered}
maint = (\frac{NC + NA + NR + DIT_{Max} + HAgg_{Max}}{5}) \\
Maintainability = \begin{cases} \text{Easy} & \text{if } maint < 0.1 \\ \text{Medium} & \text{if } 0.1 =< maint < 0.2 \\ \text{Difficult} & \text{if } maint >= 0.2 \end{cases}
\end{gathered} \tag{4}
$$

It is important to remark that the values of all the considered metrics have to be normalized in order to have the value of *maint* in the range $[0..1]$.

Figure 4 shows a fragment of the quality model defined for metamodels and containing the specification of the maintainability attribute as given in equation 4. The model consists of the definition of all the required metric providers (as listed at lines 3-8), the quality types (see lines 9-17), and the actual definition of the maintainability attribute at lines 18-21. The OCL expression implementing the definition given in equation 4 is defined in the *calculatedBy* element (see lines 27-31) and shown Listing 3.

Listing 3. OCL expressions defining the *Maintenability* attribute for metamodels

```
let x : Real =
NC_QV.type.value +  NA_QV.type.value +  NR_QV.type.value +
DITMax_QV.type.value +  HAggMax_QV.type.value)/5
in
if(x<0.2) then
    MaintainabilityQT.Easy
else if (x<0.3) then
        MaintainabilityQT.Medium
    else
        MaintainabilityQT.Difficult
    endif
endif
...
```

TABLE II
SAMPLE OF THE ANALYZED METAMODELS

Metamodel	**Maintainability**
javaforms	Easy
jtemplates	Easy
reusejava	Easy
csv	Easy
MKMCore	Easy
CPR	Easy
MavenProject	Medium
SiteConf	Medium
DiagramInterchange	Medium
CADM	Medium
BusinessDomainDsl	Medium
ConnectorGenerator	Medium
droid	Difficult
J2SE5	Difficult
Wordprocessing-MLStyles	Difficult
MiningMart	Difficult
ocl	Difficult
OntoUML	Difficult

We applied the quality model shown in Fig. 4 on a corpus of about $\approx$500 metamodels retrieved from the MDEForge repository and a sample of the obtained results is shown in Table II. Interestingly, the *OntoUML* metamodel resulted the worst metamodel to maintain and *javaforms* the best one. OntoUML is a metamodel consisting of a large number of metaclasses, it has about 30 references and 30 attributes, and it is characterized by a high number of generalizations. Javaform is a very simple metamodel, with just one metaclass and one reference. This implies that the MaxDIT and MaxHAGG are one too. The metamodel imports another metamodel, but being so simple makes it easy to maintain.

V. Conclusions And Future Work

The paper presented an extensible and generic approach to assess the quality of modelling artefacts. We proposed a tool chain enabling users to specify custom quality models, and to apply them on the artefacts being analysed in order to automatically assess their quality. We have shown the application of the proposed approach on real metamodels and ATL transformations publicly available. Thus the proposed approach permits to overcome the issues of *limited extensibility*, *artifact specificity*, and *manual assessment* affecting currently available techniques. In the future we plan to apply the approach on other artefacts e.g., editors, code generators and consider also other metrics and quality attributes, like the ones proposed in [6]. Moreover, we plan to embed the proposed approach as a new service of the MDEForge platform in order to permit the users of the system to remotely apply the quality assessment approach on the artefacts she is locally developing.

References

[1] T. Arendt, P. Stepien, and G. Taentzer, “EMF metrics: Specification and calculation of model metrics within the eclipse modeling framework,” *BENEVOL*, 2010.

[2] F. Basciani, J. Di Rocco, D. Di Ruscio, A. Di Salle, L. Iovino, and A. Pierantonio, “MDEForge: an Extensible Web-Based Modeling Platform.” *CloudMDE@MoDELS*, pp. 66–75, 2014.

[3] F. Bertoa Manuel and V. Antonio, “Quality attributes for software metamodels,” in *QAAOSE@TOOLS*, 2010.

[4] B. W. Boehm, J. R. Brown, and M. Lipow, “Quantitative evaluation of software quality,” in *ICSE*, 1976, pp. 592–605.

[5] F. Deissenboeck, E. Juergens, K. Lochmann, and S. Wagner, “Software quality models: Purposes, usage scenarios and requirements,” in *ICSE Workshop on Soft. Quality*, 2009, pp. 9–14.

[6] J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, “Mining metrics for understanding metamodel characteristics,” in *MiSE*, 2014, pp. 55–60.

[7] R. G. Dromey, “A model for software product quality,” *IEEE Trans. on Soft. Engineering*, vol. 21, no. 2, pp. 146–162, 1995.

[8] M. Genero and M. Piattini, “Empirical validation of measures for class diagram structural complexity through controlled experiments,” in *QAOOSE@ECOOP*, 2001.

[9] A. Geraci, F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, M. Yee, H. Porteous, and F. Springsteel, *IEEE standard computer dictionary: Compilation of IEEE standard computer glossaries*. IEEE Press, 1991.

[10] G. Hinkel, M. Kramer, E. Burger, M. Strittmatter, and L. Happe, “An Empirical Study on the Perception of Metamodel Quality,” in *MODELSWARD*, 2016.

[11] R. W. Hoyer, B. B. Hoyer, P. B. Crosby, W. E. Deming *et al.*, “What is quality?” *Quality Progress*, vol. 34, no. 7, p. 52, 2001.

[12] ISO, “ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model,” International Organization for Standardization, Tech. Rep., 2001.

[13] F. Jouault, F. Allilaire, J. Bzivin, and I. Kurtev, “Atl: A model transformation tool,” *Science of Computer Programming*, vol. 72, no. 12, pp. 31 – 39, 2008.

[14] S. H. Kan, *Metrics and Models in Software Quality Engineering*, 2nd ed. Addison-Wesley Longman Publishing Co., Inc., 2002.

[15] H. Kargl, M. Strommer, and M. Wimmer, “Measuring the explicitness of modeling concepts in metamodels,” in *Workshop on Model Size Metrics at MoDELS/UML*, 2006.

[16] J. J. López-Fernández, E. Guerra, and J. de Lara, “Assessing the Quality of Meta-models.” *MoDeVVa@MoDELS*, pp. 3–12, 2014.

[17] Z. Ma, X. He, and C. Liu, “Assessing the quality of metamodels.” *Frontiers of Computer Science*, vol. 7, no. 4, pp. 558–570, 2013.

[18] J. McCall, *Factors in Software Quality: Preliminary Handbook on Software Quality for an Acquisiton Manager*. General Electric, 1977.

[19] S. J. Mellor and M. Balcer, *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[20] M. Ortega, M. Pŕez, and T. Rojas, “Construction of a systemic quality model for evaluating a software product,” *Soft. Quality Journal*, vol. 11, no. 3, pp. 219–242, 2003.

[21] S. K. Rahimi, K. Lano, S. Pillay, J. Troya, and P. Van Gorp, “Evaluation of model transformation approaches for model refactoring.” *Sci. Comput. Program.*, vol. 85, pp. 5–40, 2014.

[22] D. C. Schmidt, “Guest Editor’s Introduction: Model-Driven Engineering,” *Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[23] E. Syriani and J. Gray, “Challenges for Addressing Quality Factors in Model Transformation,” in *ICST*, 2012, pp. 929–937.

[24] M. Van Amstel, C. Lange, and M. van den Brand, “Metrics for analyzing the quality of model transformations,” in *QAOOSE@ECOOP*, 2008.

[25] P. Van Roy and S. Haridi, *Concepts, Techniques, and Models of Computer Programming*, 1st ed. The MIT Press, 2004.

[26] A. Vignaga, “Metrics for measuring atl model transformations,” Tech. Rep., 2009.