Towards a Catalog of Non-Functional Requirements for Model Transformations

Soroosh Nalchigar, Rick Salay, and Marsha Chechik

Department of Computer Science, University of Toronto
{soroosh,rsalay,chechik}@cs.toronto.edu

Abstract. Model transformations play an increasingly important role in Model-Driven Engineering (MDE), and thus understanding desired non-functional requirements of model transformations and being able to determine how existing transformation languages stack up w.r.t. these is also of interest. This paper is a first step towards producing a catalog that systematically captures the transformation community's experience in developing transformations w.r.t. non-functional requirements. We survey the literature to provide a list of non-functional requirements of model transformations and find comparisons between three popular model transformation languages (ATL, QVT-Relations and AGG) according to these requirements.

1 Introduction

Model Driven Engineering (MDE) is a software engineering discipline in which models are the primary artifacts and play a central role throughout the entire development process [25,27]. Model transformations are the core MDE mechanism for building software from design to code, and hence have a significant impact on the software development process [8]. They are used for different reasons and intents, e.g., to extract different views from a model (query), add or remove detail (refinement or abstraction), generate code from model (synthesis) [1].

Since model transformations play a critical role in the MDE process, their non-functional requirements (NFRs) are of great importance as well. Yet, existing research only addresses transformation NFRs indirectly. The NFRs that a transformation language or tool should satisfy has been studied and these have been proposed as a way of selecting a suitable language or tool [4,25,35]. Another line of research looks at the quality attributes and metrics for the model transformation artifact itself (i.e., the code) with the aim of developing quality assurance techniques for model transformations [3,27,41].

Since a transformation is a combination of artifact with a language, the characteristics of a transformation are contributed to by the characteristics of both the artifact and the language. Thus, these two lines of research must be combined when considering the NFRs of transformations. We address this in the current paper, making the following contributions. First, we normalize the findings from both these lines of research to create a unified set of NFRs for transformations reflecting the contributions of the artifact and language aspects. Next, we apply these NFRs to compare three popular transformation languages (ATL, QVT-Relations and AGG) based on evidence from the literature. We believe that this paper is a first step towards producing a catalog that systematically captures the transformation community's experience in developing transformations w.r.t. non-functional requirements.

The rest of this paper is structured as follows. Sec. 2 establishes a list of non-functional requirements. Sec. 3 compares three existing model transformation languages and examines how they contribute to non-functional requirements. We summarize our results and discuss directions for future work in Sec. 4.

2 NFRs for Model Transformations

In this section, we build a list of non-functional requirements (NFRs) for model transformations. Our methodology for doing so is by reviewing previous published research on this topic and unifying the results. The list of papers we reviewed was constructed by running Google Scholar using the keywords "model transformation", "quality", "non-functional", and "requirements". Having the results sorted by relevance, we examined the titles and abstracts, choosing papers that have proposed quality measures and NFRs of model transformations, while excluding those dealing with transforming requirements in MDE since they are out of the scope of our study. This resulted in 10 papers that we survey below.

Literature Review. Amstel et al. [3, 41] proposed a set of eight quality attributes for model transformation artifacts: *understandability*, *modifiability*, *reusability*, *reuse*, *modularity*, *completeness*, *consistency*, and *conciseness.* The authors also proposed a set of metrics for assessing them, comprised of *size* (e.g., the number of functions), *function* (e.g., the number of equations and conditions per function), *module* (e.g., number of library modules), and *consistency* (e.g., number of variables per type). Similar metrics have been proposed by Vignaga [42] to assess the quality of ATL model transformations. Amrani et al. [1] presented a catalog of model transformation intents and mentioned *readability* as a property of model transformations. Syriani and Gray [37] proposed a categorization of quality attributes for model transformations that included *correctness*, *reusability*, *efficiency*, *reliability*, *maintainability*, and *interoperability.*

Mens and Gorp [25] differentiated between functional and quality requirements of model transformation languages and tools and proposed *usability and usefulness*, *verbosity*, *conciseness*, *scalability*, *extensibility*, *interoperability*, *acceptability* and *standardization* as the main non-functional requirements. Mens et al. [26] continued this line of research and examined how well graph transformation languages satisfy these quality requirements. Aziz [4] evaluated quality of four model transformation technologies (ATL, IBM transformations, Acceleo, and Java APIs) for model-to-model and model-to-text transformations in a case study in Ericsson AB. His quality model included *usability* (comprised of *understandability*, *learnability*, and *operability*), *maintainability* (comprised of *analyzability* and *changeability*), *functionality* (comprised of *suitability* and *accuracy*), and *scalability.* Sendall and Kozaczynski [35] proposed a set of desirable characteristics for a model transformation language including *usability*, *ease of understanding*, *ease of modification*, *conciseness*, *acceptability*, *composition and reuse*,

Non-functional Requirement	**Language contribution**	**Artifact contribution**	**Sources**
Understandability (**UN**)	The extent to which a model transformation language is easy to understand, use, and learn by developers.	The amount of effort required to understand model transformation code.	[1, 3, 4, 27, 35, 41, 42]
Performance and Scalability (**PS**)	Ability of the language or tool to cope with large and complex transformations or transformations of large and complex software models without sacrificing performance.	The extent to which a transformation artifact works efficiently and uses few resources to execute.	[4, 25, 26, 37]
Extensibility (**EX**)	The ease with which the tool can be extended with new functionality.	N/A	[25–27]
Interoperability (**IN**)	The ease with which the tool can be integrated with other tools used within the (model-driven) software engineering process.	N/A	[25–27, 37]
Verbosity (**VE**)	Amount of syntactic sugar in the transformation language for frequently used syntactic constructs.	Amount of syntactic sugar and other additional constructs in the artifact.	[25, 26]
Conciseness (**CO**)	Absence of syntactic sugar in the transformation language.	The extent to which a model transformation does not include superfluous information, e.g., code clones, unnecessary function parameters).	[25, 26, 35, 41]
Modifiability (**MOF**)	N/A	The extent to which a model transformation artifact can be changed and adapted to provide different or additional functionality.	[3,4,27,35,37, 41, 42]
Visualization (**VI**)	Whether the transformation technology provides visual specifications of transformation.	N/A	[4, 35]
Modularity (**MDL**)	The ability of the language to support modularity.	The extent to which a model transformation is systematically structured (every model in a model transformation has its own purpose).	[3, 27, 41, 42]
Standardization (**ST**)	Whether the transformation tool is compliant to all relevant standards such as XML, UML, and MOF (Meta Object Facility).	Whether the transformation artifact conforms to standard import/export formats for models.	[25, 26, 37]
Reusability (**REY**)	Availability of a reuse mechanisms within the language.	The extent to which (a part of) a model transformation can be reused by other model transformations (as-is reuse).	[3, 27, 35, 37, 41, 42]
Reuse (**RE**)	Availability of reuse mechanisms within the language.	The extent to which a model transformation reuses parts of other model transformations.	[4, 27, 35, 41, 42]
Usability and Usefulness (**UU**)	The general utility and ability of the language to serve practical purposes and be intuitive and efficient to use.	N/A	[25, 26, 35]
Acceptability (**AC**)	The extent to which a tool is accepted by the user community.	N/A	[25, 26, 35]

Table 1. Summary of model transformation NFRs in previous studies.

and *visual representation*. Mohagheghi and Aagedal [27] presented *comprehensibility*, *modifiability*, *modularity*, *reusability*, *extensibility* and *interoperability* as quality goals in MDE.

Towards a List of Non-Functional Requirements. In order to combine the NFRs and provide a single comprehensive list, we identified the equivalent attributes from different works and unified them into a single attribute. For example, *modifiability* from [3,27,41], *changeability* from [4], *maintainability* from [37], and *ease of modification* from [35]) are considered to be a single attribute, called *modifiability* in our list. Then, we removed *completeness*, *consistency*, *correctness*, *accuracy*, and *reliability* because we believe these are functional rather than NFRs (and can have boolean valuations).

Based on this review, Table 1 summarizes and defines a set of non-functional requirements of model transformations (first column), lists the contributions of the language and artifact aspects (second and third columns, respectively), and cross-references them with the relevant citations (last column).

Obviously, the current list of NFRs could be extended by new items, identified in future papers.

3 Application: NFRs of Three Model Transformation Languages

In this section, we apply the NFRs from the list established in Sec. 2 to three popular model transformation languages: ATLAS Transformation Language (ATL) [19], Attributed Graph Grammar (AGG) [38], and QVT-Relations [21,28]. In choosing these languages for comparison, we considered factors such popularity of the languages as well as the availability of previous studies involving their NFRs.

3.1 Transformation Languages

Figure 1 shows a part of a simple transformation of a class schema model to a relational database model implemented in these languages.

ATL is a hybrid (mix of declarative and imperative constructs) model transformation language developed as a part of the ATLAS Model Management Architecture (AMMA) platform. Figure 1(a) shows the header of the transformation where the source and target models are declared. It also indicates a transformation rule that creates a table instance for each of the class instances.

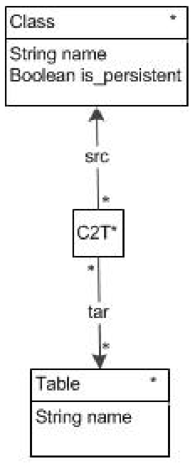
AGG is a general development environment for attributed graph transformations, supporting the algebraic approach. Figure 1(b) shows part of the type graph for a class diagram (the top) and a relational database model (the bottom), as well as helper structures that hold the correspondences between the elements of different type graphs (the middle).

QVT is the OMG standard language for specifying model transformations and includes three sublanguages: QVT-Relations (surveyed here and referred to as **QVT-R**), QVT-Operational and QVT-Core. Figure 1(c) shows a part of the aforementioned transformation expressed in QVT-R. In this example, each relation represents a mapping, whereas a domain declares a pattern and is bound to a model (e.g., UML).

```
module Class2Relational;
create OUT :
SimpleRelational
from IN :
SimpleClass;

rule Class2Table {
  from
    c : Class!Class
  to
    out : Relational!Table (
      name <- c.name,
      col <- Sequence {key}->
         union(c.attr->select(e
        | not e.multiValued)),
      key <- Set {key}
    ),
  key : Relational!Column (
    name <- 'objectId',
    type <- thisModule.
            objectIdType
  )
}
...
```

(a)



(b)

```
transformation umlRdbms (
  uml : SimpleUML,
  rdbms : SimpleRDBMS) {
  key Table (name, schema);
  key Column (name, table);

  top relation ClassToTable {
    domain uml c:Class {
      package = p:Package {},
      isPersistent = true,
      name = cn
    }

    domain rdbms t:Table {
      schema = s:Schema {},
      name = cn,
      cols = cl:Column {
      name=cn+_tid,
      type=NUMBER},
      pkey = cl
    }
    ...
  }
```

(c)

Fig. 1. Part of a sample transformation: (a) in ATL [17]; (b) in AGG [39]; and (c) in QVT-R [9].

3.2 Methodology

We reviewed the literature, searching for theoretical and user comments as well as empirical studies that compared the three languages w.r.t. each of the NFRs.

For each NFR, existing papers usually provide pairwise relative rankings rather than absolute valuations of the NFR. Thus, our rankings between languages should also be interpreted as relative. Note also that we did not perform empirical studies ourselves, and our comparisons are only based on published work, not on our own experience. Finally, we did not include *usability / usefulness* and *acceptability* into our comparisons because we specifically chose popular languages which, by definition, score high on these characteristics.

Our results are summarized in Table 2. In this table, we use a qualitative scale where three stars ($***$) represent the best and one star ($*$) – the worst performance w.r.t each quality attribute. Also, in the case of binary evaluations, "true" is mapped to $***$ and "false" – to $*$. We summarize the source of our evidence using the letters *A*-*D*: *A* indicates an empirical experiment or a case study; *B* – information coming from the language description; *C* – a qualitative description; and *D* – all other sources.

3.3 Results

In the rest of the section, we discuss and compare the languages. w.r.t. each of the non-functional attributes.

Understandability: For this attribute, we looked for the relevant literature than has compared these languages w.r.t. the overall size of the model transformation in terms of the number of lines of code and the transformation complexity [41]. [14] reported that AGG is more compact than ATL w.r.t. the

Languages	UN	PS	EX	IN	VE	CO	MOF	VI	MDL	ST	REY	RE
ATL	$\ast\ast(A)$	$\ast\ast\ast(A)$	$\ast\ast\ast(B)$	$\ast\ast\ast(C)$	$\ast\ast\ast(A)$	$\ast(A)$	$\ast(A)$	$\ast\ast(B)$	$\ast\ast\ast(A)$	$\ast\ast\ast(B)$	$\ast\ast\ast(A)$	$\ast\ast\ast(A)$
QVT-R	$\ast(A)$	$\ast(A)$	$\ast\ast(B)$	$\ast\ast\ast(C)$	$\ast\ast\ast(C)$	$\ast(C)$	$\ast(D)$	$\ast\ast(B)$	$\ast\ast\ast(A)$	$\ast\ast\ast(C)$	$\ast\ast\ast(C)$	$\ast\ast\ast(C)$
AGG	$\ast\ast\ast(A)$	$\ast(C)$	$\ast\ast(C)$	$\ast(C)$	$\ast(C)$	$\ast\ast\ast(C)$	$\ast\ast\ast(D)$	$\ast\ast\ast(B)$	$\ast(B)$	$\ast\ast\ast(B)$	$\ast(C)$	$\ast(C)$

Table 2. Summary of comparison of languages with regarding to non-functional requirements. See Table 1 for abbreviations.

amount of code that a programmer needs to write in order to make an executable solution. [30] reported that QVT-R transformations are larger than ATL in terms of number of line of code and also have the larger complexity in terms of the number of recursive calls. According to [4], ATL includes a mix of imperative and declarative constructs; while it makes the language powerful and compact, it decreases language understandability. Based on these comparisons, we infer that AGG is better than ATL, and ATL is better than QVT-R w.r.t. understandability (column **UN** in Table 2).

Performance and scalability: According to [2], the ATL language is generally faster than QVT-R: their experiment of increasing either the size of the complexity of the input models yielded ATL running times to be more than five times faster than their QVT-R counterparts. [36] states that ATL is capable of managing complex models because of its imperative language constructs and the use of helper functions. Also, according to [26], graph transformations are generally considered to generate inefficient programs. Hence, we rank these languages w.r.t. performance and scalability as ATL (best), QVT-R, and AGG (worst) (column **PS** in Table 2).

Extensibility: To assess this characteristics, we looked for literature that provided evidence on successful extensions to our three transformation languages and then assigned boolean values to the languages w.r.t. this attribute. Randak et al. [31] extended ATL with new keywords to natively support UML profiles in transformations. The preprocessor then translates these keywords into standard ATL syntax by a higher-order transformation. Mens et al. [26] argue that the AGG tool is extensible in the sense that its internal graph transformation engine, implemented in Java, can be extended freely to cover a variety of different applications. [29] extended QVT-R and integrated it with constraint programming for specification of attribute values in target models. Also, [10] extended QVT-R to embed information about system variants and their impact on the non-functional properties of the system being developed. Finally, several extensions to ATL via virtual machines have been recently reported: [34], for adding bidirectionality, and Eclectic [33] and EMFTVM [43] for richer bytecode. These support the finding that ATL is more extensible than either QVT-R or AGG (whom we deem equally extensible) – column **EX** in Table 2.

Interoperability: Our comparison of languages w.r.t. interoperability is similar to the one we conducted for extensibility. [20] reports that languages based on the graph transformation paradigm (e.g., AGG) employ graph patterns and it is not clear how OCL-based queries are translated to graph patterns and vice versa. [6] describes the procedure for making AGG and Ecore mutually interoperable. Interoperability of ATL and QVT-R is discussed and shown in [18]. Based on these, we infer "false", "true", and "true" for the interoperability of AGG, QVT-R, and ATL, respectively (column **IN** in Table 2).

Verbosity and Conciseness: We consider conciseness to be the inverse of verbosity. [24] states that QVT-R rely only on the textual language OCL for model querying, and this leads to verbose and complicated OCL expressions. Also, according to [26], graph transformation languages (e.g., AGG) are concise. In addition, *understandability*-related references, e.g., [14], also support the argument that ATL is more verbose than AGG. Based on these, we infer that ATL and QVT-R are more verbose (less concise) than AGG. We were unable to find a direct comparison between ATL and QVT-R regarding this attribute, so we give these languages the same valuation (columns **VE** and **CO** in Table 2).

Modifiability: [13] states that ATL reduces modifiability of model transformations because it implements the source pattern using multiple rules and helpers, all of which need to be updated even for a minor constraint change in source pattern definition. [40] mentions that the manipulation of transformation models in ATL is complex because of the inherent complexity of programming language metamodels. Modifiability can be seen to be positively related to understandability. Hence, we infer that AGG is better than ATL and QVT-R with regarding to modifiability, whereas the direct comparison between ATL and QVT-R is not available (column **MOF** in Table 2).

Visualization: AGG provides graphical languages and a development environment to define model-to-model transformations [11]. Graph transformations are defined over metamodel elements and visualized with a generic layout, called abstract syntax, where nodes are visualized as rectangles, and edges – as directed arrows [14]. Various graph layout options are offered for tuning the visualization [7]. AGG can visualize simulations as well. While abstract syntax of a given modeling language is less familiar than its concrete counterparts to the developers [9,14], the visualization and simulation capabilities of AGG make it intuitive and easy to understand [7]. QVT-R has a graphical notation, but its graphical editors are just recently being proposed [23] and further tool support is lacking. Graphical editors for transformations with ATL exist, e.g., ATLFlow [44]. ATLFlow is able to describe the structure of a transformation and execute it, but it does not support conditional branches or composite transformations [32]. We conclude that AGG's graphical support and visual representation are better than ATL's or QVT-R' (column **VI** in Table 2).

Modularity: We use boolean values to evaluate this NFR attribute w.r.t. the question "does the language include constructs to support modularity". ATL supports modularity and allows packaging rules into modules. A module can import another module to access its content [9, 14, 22]. QVT-R supports modularity as well [30]. We could not find any evidence of modularity support in AGG. Hence, we assign "false", "true", and "true" to the modularity aspect of AGG, QVT-R and ATL, respectively (column **MDL** in Table 2).

Standardization: The two standards for graph transformation languages (GXL – an exchange format for graphs, and GTXL – an exchange format for graph transformations) are supported by AGG. It also supports XML and can be used for refactoring UML models [11, 12, 26]. Moreover, it is compliant with the relevant standards such as UML, MOF, and XML. ATL can be used to bridge XML, Grafcet, Petri net, and PNML [16], and an example of the MOF-to-UML transformation is given in [15]. Finally, QVT languages are compatible with MOF, UML, and OCL [21]. We thus infer that all the three languages have proper compliance with the relevant standards (column **ST** in Table 2).

Reusability and Reuse: We interpret support for *both* reusability and reuse as "does the language have constructs that support reuse", and thus assign boolean values to the three languages w.r.t. this attribute. AGG has no explicit reuse mechanisms such as rule inheritance, derivation, extension and specialization [9, 26]. ATL's support of Higher-Order Transformations can be treated as enabling reuse since they allow to analyze, produce and manipulate other model transformations [5, 40]. The specification of QVT-R [21] indicates that its architecture supports reuse of existing libraries and allows plugging-in and executing external code. Thus, we evaluate the reuse support in the three languages as "false", "true", and "true", respectively (column **REY** in Table 2).

We end this section with some remarks. First, the comparisons in this paper use a relative scale, with qualitative rather than absolute values, so '$***$' is not necessarily "good", just "better", and '$*$' is not necessarily "bad", just "worse". And obviously, our comparison is limited to only three languages, and we do not claim that our list of NFRs is complete or comprehensive.

4 Conclusion

In this paper, we reviewed the non-functional attributes of model transformations as described in previous works, and created their comprehensive list. We then reviewed the literature to compare three popular model transformation languages w.r.t. this list, resulting in a non-functional requirements catalog. We expect this catalog to be used by language designers to help improve their language, as well as by developers aiming to choose the right transformation language for their job. We also hope that the community will join this effort both by growing the list of non-functional qualities and by adding other transformation languages to the catalog. Future work can extend the proposed list of non-functional properties by dividing each attribute into more quantifiable chunks and proposing metrics for effectively measuring them. Another line of future work can be to propose methods and mechanisms to assist software developers in finding the most suitable language and tool based on their NFRs.

References

1. M. Amrani, J. Dingel, L. Lambers, L. Lucio, R. Salay, G. Selim, E. Syriani, and M. Wimmer. Towards a Model Transformation Intent Catalog. In *Proc. of AMT'12 (MODELS'12 Wksp.)*, 2012.

2. M. Amstel, S. Bosems, I. Kurtev, and L. Ferreira Pires. Performance in Model Transformations: Experiments with ATL and QVT. In *Proc. of ICMT'11*, volume 6707 of *LNCS*, pages 198–212, 2011.
3. M. F. v. Amstel. The Right Tool for the Right Job: Assessing Model Transformation Quality. In *Proc. of COMPSACW'10*, pages 69–74, 2010.
4. K. M. A. Aziz. *Evaluating Model Transformation Technologies: An Exploratory Case Study*. PhD thesis, Chalmers University of Technology, 2011.
5. J. Bézivin, E. Breton, G. Dupé, and P. Valduriez. The ATL Transformation-Based Model Management Framework. Technical report, Université de Nantes, 2003.
6. E. Biermann, C. Ermel, L. Lambers, U. Prange, O. Runge, and G. Taentzer. Introduction to AGG and EMF Tiger by Modeling a Conference Scheduling System. *In J. STTT*, 12(3-4):245–261, 2010.
7. E. Biermann, C. Ermel, L. Lambers, U. Prange, O. Runge, and G. Taentzer. Introduction to AGG and EMF Tiger by Modeling a Conference Scheduling System. *Int. J. Softw. Tools Technol. Transf.*, 12(3-4):245–261, 2010.
8. E. Brottier, F. Fleurey, J. Steel, B. Baudry, and Y. Le Traon. Metamodel-Based Test Generation for Model Transformations: an Algorithm and a Tool. In *Proc. of ISSRE'06*, pages 85–94, 2006.
9. K. Czarnecki and S. Helsen. Feature-Based Survey of Model Transformation Approaches. *IBM Syst. J.*, 45(3):621–645, 2006.
10. M. Drago, C. Ghezzi, and R. Mirandola. A Quality Driven Extension to the QVT-relations Transformation Language. *Computer Science - Research and Development*, pages 1–20, 2011.
11. H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. Implementation of Typed Attributed Graph Transformation by AGG. *Fundamentals of Algebraic Graph Transformation*, pages 305–323, 2006.
12. A. Folli and T. Mens. Refactoring of UML models using AGG. *ECEASST*, 8, 2007.
13. A. Göknil, N. Topaloglu, and K. van den Berg. Operation Composition in Model Transformations with Complex Source Patterns. Technical Report TR-CTI, Centre for Telematics and Information Technology, University of Twente, 2008.
14. R. Gronmo, B. Moller-Pedersen, and G. Olsen. Comparison of Three Model Transformation Languages. In *Proc. of ECMDA-FA'09*, volume 5562 of *LNCS*, pages 2–17, 2009.
15. A. group. The MOF to UML ATL transformation. Technical report, LINA & INRIA, September 2005.
16. P. Guyard. ATL Transformation Example: Bridging Grafcet, Petri net, PNML and XML. Technical report, INRIA, August 2005.
17. INRIA. ATL Transformation Example: Class to Relational, 2005. `http://www.eclipse.org/atl/atlTransformations/Class2Relational/ExampleClass2Relational[v00.01].pdf`. Last Accessed July 2013.
18. F. Jouault and I. Kurtev. On the Architectural Alignment of ATL and QVT. In *Proc. of SAC'06*, pages 1188–1195, 2006.
19. F. Jouault and I. Kurtev. Transforming Models with ATL. In *Proc. of MODELS'05*, pages 128–138, 2006.
20. F. Jouault and I. Kurtev. On the Interoperability of Model-to-Model Transformation Languages. *Science of Computer Programming*, 68(3):114 – 137, 2007.
21. I. Kurtev. State of the Art of QVT: A Model Transformation Language Standard. In *Proc. of AGTIVE'07*, volume 5088 of *LNCS*, pages 377–393, 2008.
22. I. Kurtev, K. Van Den Berg, and F. Jouault. Evaluation of Rule-Based Modularization in Model Transformation Languages Illustrated with ATL. In *Proc. of SAC'06*, pages 1202–1209, 2006.

23. D. Li, X. Li, and V. Stolz. QVT-based Model Transformation using XSLT. *SIGSOFT SEN*, 36(1):1–8, 2011.
24. D. Li, X. Li, and V. Stolz. Model Querying with Graphical Notation of QVT Relations. *SIGSOFT Softw. Eng. Notes*, 37(4):1–8, July 2012.
25. T. Mens and P. Van Gorp. A Taxonomy of Model Transformation. *El. Notes Theor. Comp. Sci.*, 152:125–142, 2006.
26. T. Mens, P. Van Gorp, D. Varró, and G. Karsai. Applying a Model Transformation Taxonomy to Graph Transformation Technology. *El. Notes Theor. Comput. Sci.*, 152:143–159, 2006.
27. P. Mohagheghi and J. Aagedal. Evaluating Quality in Model-Driven Engineering. In *Proc. of MiSE'07*, pages 6–15, 2007.
28. OMG. Meta Object Facility (MOF) 2.0 Query / View / Transformation. Technical report, OMG Technical Report, 2011.
29. A. Petter, A. Behring, and M. Mhlhuser. Solving Constraints in Model Transformations. In *Proc. of ICMT'09*, volume 5563 of *LNCS*, pages 132–147, 2009.
30. S. Rahimi and K. Lano. Integrating Goal-Oriented Measurement for Evaluation of Model Transformation. In *Proc. of CSSE'11*, pages 129–134, 2011.
31. A. Randak, S. Martínez, and M. Wimmer. Extending ATL for Native UML Profle Support: An Experience Report. In *Proc. of CEUR'11 Workshop*, 2011.
32. J. E. Rivera, D. Ruiz-Gonzalez, F. Lopez-Romero, J. Bautista, and A. Vallecillo. Orchestrating ATL Model Transformations. In *Proc. of MtATL'09*, pages 34–46, 2009.
33. J. Sánchez Cuadrado. Towards a Family of Model Transformation Languages. In *Proc. of ICMT'12, LNCS*, volume 7307, pages 176–191, 2012.
34. I. Sasano, Z. Hu, S. Hidaka, K. Inaba, H. Kato, and K. Nakano. Toward Bidirectionalization of ATL with Groundtram. In *Proc. of ICMT'11*, pages 138–151, 2011.
35. S. Sendall and W. Kozaczynski. Model Transformation: the Heart and Soul of Model-Driven Software Development. *IEEE Software*, 20(5):42–45, 2003.
36. M. Stephan and A. Stevenson. A Comparative Look at Model Transformation Languages. Technical report, Soft. Tech. Lab, Queen's University, 2009.
37. E. Syriani and J. Gray. Challenges for Addressing Quality Factors in Model Transformation. In *Proc. of ICST'12*, pages 929 –937, April 2012.
38. G. Taentzer. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In *Proc. of AGTIVE'03*, volume 3062 of *LNCS*, pages 446–453, 2004.
39. G. Taentzer, K. Ehrig, E. Guerra, J. De Lara, L. Lengyel, T. Levendovszky, U. Prange, D. Varró, and S. Varró-Gyapay. Model Transformation by Graph Transformation: A Comparative Study. In *MODELS'05 Wksp. MT in Practice*, 2005.
40. M. Tisi, J. Cabot, and F. Jouault. Improving Higher-Order Transformations Support in ATL. In *Proc. of ICMT'10*, pages 215–229, 2010.
41. M. F. van Amstel, C. F. J. Lange, and M. G. J. van den Brand. Metrics for Analyzing the Quality of Model Transformations. In *ECOOP'08 Wksp. on Quantitative Approaches to OO SE*, 2008.
42. A. Vignaga. Metrics for Measuring ATL Model Transformations. Technical report, Universidad de Chile, 2009.
43. D. Wagelaar, M. Tisi, J. Cabot, and F. Jouault. Towards a General Composition Semantics for Rule-Based Model Transformation. In *Proc. of MODELS'11*, pages 623–637, 2011.
44. U. Zeidler. ATLflow Plugin. `http://opensource.urszeidler.de/ATLflow/`. Last Accessed Dec. 2012.