Contrasting Dedicated Model Transformation Languages vs. General Purpose Languages: A Historical Perspective on ATL vs. Java based on Complexity and Size – Extended Abstract

Stefan Höppner[1], Timo Kehrer[2], Matthias Tichy[1]

Abstract:

Model transformations are one key concept of model-driven engineering, and model transformation languages (MTLs) emerged with its popularity about 15 to 20 years ago. MTLs claim to ease model transformation development by abstracting from recurring transformation aspects and hiding complex semantics behind simple and intuitive syntax. Nonetheless, MTLs are rarely adopted in practice, there is still no empirical evidence for the claim of easier development, and the argument of abstraction deserves a fresh look in the light of modern general-purpose languages (GPLs) which have undergone a significant evolution in the last two decades. In our SoSyM paper [HKT21], we report on a study in which we compare the complexity and size of model transformations written in three different languages, namely (i) the Atlas Transformation Language (ATL), (ii) Java SE5 (2004-2009), and (iii) Java SE14 (2020); the Java transformations are derived from an ATL specification using a translation schema we developed. Based on the results of these comparisons, we discuss the concrete advancements in newer Java versions. We also discuss to which extent new language advancements justify writing transformations in a GPL rather than a dedicated MTL. We further indicate potential avenues for future research on the comparison of MTLs and GPLs.

Keywords: ATL; Java; Model transformations; Model transformation language; General purpose language; Comparison; MTL vs. GPL; Historical Perspecitve; Complexity Measure; Size Measure

In the literature, many advantages are ascribed to model transformation languages, such as better analysability, comprehensibility or expressiveness [GTG21]. Nowadays, however, such claims have two main flaws. First, there is a lack of actual evidence to have confidence in their genuineness [GTG21]. And second, most of these claims emerged around 15 years ago when the first model transformation languages were introduced. From this follows the presumption that transformations can just as well be written in a modern GPL. This presumption is confirmed by a community discussion on the future of model transformation languages [BCG19], and partially by an empirical study conducted by Hebig et al. [He18]. The presumption is mainly rooted in the idea that new language features allow developers to heavily reduce boilerplate code that MTLs claim to abstract away from. To validate and

[1] Ulm University, Institute for Software Engineering and Compiler Construction, James-Franck-Ring 1, 89081 Ulm, Germany [stefan.hoeppner|matthias.tichy]@uni-ulm.de

[2] Humboldt University Berlin, Institute for Informatics, Unter den Linden 6, 10099, Germany timo.kehrer@informatik.hu-berlin.de

better understand this argumentation, we elected to compare 12 transformations written in the Atlas Transformation Language (ATL), one of the most widely known MTLs, with the same transformations written in a recent version of Java (Java SE14), as well as an older version (Java SE5), that was recent when ATL was introduced. The **goal** of our SoSyM paper [HKT21] was to (i) analyse how much transformation code, written in Java, has improved over the years and (ii) contextualise these improvements by relating them to ATL code. To do so, we opted to use both size and complexity measures to gain insights into the makeup of transformation code for the discussion.

Our results show that new features introduced in Java since 2006 help to significantly reduce the complexity and lines of code of transformations written in Java. However, they do not reduce the number of words required to write the same transformations. This suggests an ability to express more information dense code in newer Java versions. Transformations in newer Java versions can also be written in a more data driven way that more closely resembles the way they are defined in MTLs. We also showed that, while the overall complexity of transformations is reduced, the distribution of how much of that complexity stems from code that implements functionality that ATL and other MTLs can hide from the developer stays about the same. This observation is further supported by the analysis of code size distribution. Here, we found that while large parts of the transformation classes relate to the transformation process itself, within those parts there is still significant overhead from tracing as well as general supplemental code required for the transformations to work. It follows, that while the overall complexity is reduced, the overhead entailed by using a general purpose language for writing model transformations is still present.

Overall we find that more recent Java versions make development of transformations easier because less work is required to set up a working transformation, and the creation of output elements and the assignment of their attributes are now a more prominent aspect within the code. From our results and experience with this and other projects, we also conclude that general purpose languages are most suitable for transformations where little to no tracing is required because the overhead associated with this transformation aspect is the most prominent one and holds the most potential for errors. However, we also believe that advanced features such as property preservation verification or bidirectional and incremental transformation development cannot currently be implemented with justifiable effort in a general purpose language.

Data Availability

All transformation and analysis code involved in our work is publicly accessible on the OPARU system of our university [HTK21].

Bibliography

[BCG19] Burgueño, Loli; Cabot, Jordi; Gerard, Sebastien: The Future of Model Transformation Languages: An Open Community Discussion. Journal of Object Technology, 18(3):7:1–11, July 2019. The 12th International Conference on Model Transformations.

[GTG21] Götz, Stefan; Tichy, Matthias; Groner, Raffaela: Claimed advantages and disadvantages of (dedicated) model transformation languages: a systematic literature review. Software and Systems Modeling, 20(2):469–503, 2021.

[He18] Hebig, Regina; Seidl, Christoph; Berger, Thorsten; Pedersen, John Kook; Wasowski, Andrzej: Model Transformation Languages Under a Magnifying Glass: A Controlled Experiment with Xtend, ATL, and QVT. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA, 2018.

[HKT21] Höppner, Stefan; Kehrer, Timo; Tichy, Matthias: Contrasting dedicated model transformation languages versus general purpose languages: a historical perspective on ATL versus Java based on complexity and size. Software and Systems Modeling, 2021.

[HTK21] Höppner, Stefan; Tichy, Matthias; Kehrer, Timo: , Contrasting Dedicated Model Transformation Languages vs. General Purpose Languages: A Historical Perspective on ATL vs. Java based on Complexity and Size: Supplementary Materials, 2021.