[GLR17] T. Greifenberg, M. Look, B. Rumpe:
Visualizing MDD Projects.
In: Software Engineering Conference (SE'17), LNI, pp. 101-104. Bonner Köllen Verlag, 2017.
www.se-rwth.de/publications/

Visualizing MDD Projects

Timo Greifenberg[1], Markus Look[1], Bernhard Rumpe[1]

Abstract: Visualizing information enables humans to capture, understand, and analyze them in an efficient way. Tools providing such visualization possibilities are omnipresent in software development processes and still subject to current research. While visualization is widely tried in classical software development, the application to MDD is much less common, but still desperately needed. In this paper we outline an integrated and pervasive visualization approach for artifacts and software engineering tools as well as their static and dynamic dependencies by employing, extending, and combining well established visualization approaches to the MDD domain.

Keywords: Software Engineering, Model-driven Software Engineering, Visualization, Artifacts, Generative Software Engineering

1 Introduction

Visualizing information [BS03] enables humans to capture, understand, and analyze them in an efficient way. Thus, tools providing such visualization possibilities are omnipresent in classical software development processes and still subject to current research. These tools help to understand the development of today's complex software systems. In contrast, Model-driven development (MDD) lacks this integrated visualization tool support. MDD aims at employing models as primary development artifacts to abstract from technological and reoccurring details. MDD requires a partly automated development process, where software engineering tools such model transformations and code generators perform model-to-model and model-to-text transformations [CH03] in order to generate the source code of a complex software system. It is our believe, that due to the large amount of involved artifacts, their dependencies, the amount of employed software engineering tools, the high degree of automation, and new roles in the process with different visualization needs, it is crucial to provide an integrated visualization tool landscape for MDD projects. In this paper, we outline an idea of employing, extending and combining well established tools and techniques to the MDD domain. By doing so, an integrated and pervasive visualization for artifacts, software engineering tools, and their different kinds of dependencies is created.

We will start with notion of MDD Projects, by presenting the involved artifacts, tools and dependencies between them in Section 2. Built upon this, we will show how we imagine related tools being utilized and combined in order to visualize MDD projects coherently in Section 3. Section 4 concludes our idea.

2 MontiCore-based MDE/MDD Projects

Before explaining how elements and relations of MDD projects could be visualized, our notion of MDD projects is presented. We concentrate on the development phase of the project and especially on the tools and artifacts involved. This means, that we are leaving aside all other terms, which can be regarded as part of a project, such as roles, manual executed activities, informal requirements and project management activities. Figure 1 sketches our view on MDD projects. We use the language workbench MontiCore [Gr08]

[1] RWTH Aachen University, Chair of Software Engineering, Ahornstr. 55, 52074 Aachen, www.se-rwth.de

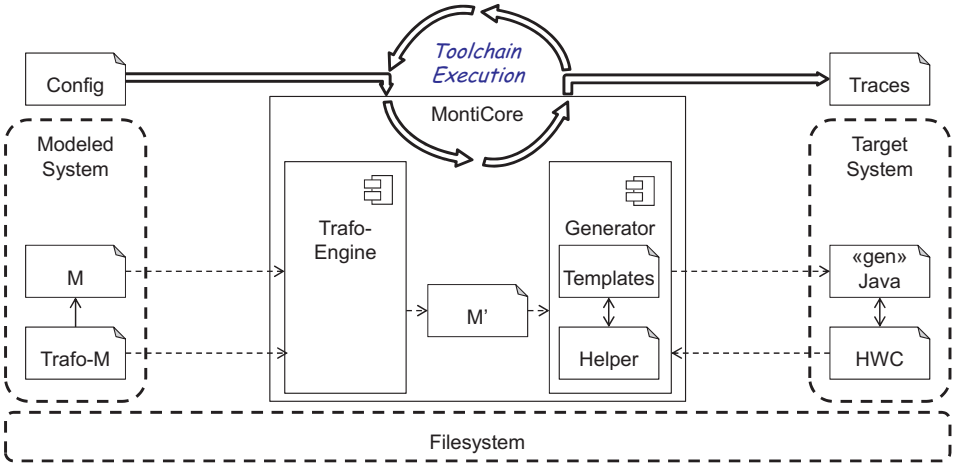

Figure 1: MDD projects with its tools and artifacts.

to enable our MDD processes efficiently. Models, shown as M, serve as primary development artifacts, capturing requirements and specification. In addition, transformation models, shown as $Trafo\text{-}M$, are processed by the MontiCore transformation engine to perform model-to-model transformations for the input models. Generators use (transformed) models, shown as M', as input to generate parts of the target system. Note, that Figure 1 only shows two tools (*Trafo-Engine* and *Generator*), while in general, an arbitrary number of tools form MDD toolchains. In addition, the system can consist of handwritten code, shown as HWC, which has relations to the generated files. Furthermore, the same HWC is respected and reacted upon during generation by the generator. The code generator is composed of templates, which can execute other templates or call helper functions, written in Java. Thus, we consider models, transformation models and handwritten code files, as primary input artifacts of an MDD process. Templates and Java artifacts providing helper functionality form the code generator and Java artifacts including HWC files are the primary output. HWC is regarded as input and as output artifact. Furthermore, the toolchain requires configuration, which is also captured in an artifact, shown as $Config$. During execution of the toolchain artifacts containing trace information, shown as $Traces$, are created as a side product.

This together leads to a plethora of dependencies within an MDD process. In general we consider static and dynamic dependencies. Static dependencies are relations that can be determined by analyzing the involved artifacts without executing the toolchain while dynamic dependencies are relations that can only be observed when executing the toolchain. Static dependencies consist between input artifacts (M and $Trafo\text{-}M$), code generator artifacts ($Templates$ and $Helpers$) as well as output artifacts (generated and handwritten code). Additionally, dynamic dependencies such as templates that are executed by other templates and tools that create files occur during toolchain execution. Moreover artifact-based architectures can be defined within the MDD project. Dependencies between architectural structures, composed of artifacts, can be derived from the dependencies of their artifacts. This shows, that there are other kinds of elements and dependencies to be con-

sidered than in classic software development, which motivates the integrated visualization approach introduced next.

3 Visualizing MDD Projects

In this section, we envision an integrated visualization approach for MDD projects. The different elements and dependencies of MDD projects are mapped to existing visualization tools. Moreover, the approach describes the interaction between the different perspectives leading to an integrated and preservative visualization.

Toolchains: The visualization of toolchains is the entry point of our integrated visualization approach. Here, the configuration of the MDD project's toolchain is displayed. Tools are connected by their expected input and output artifacts and a tool could be applied more than once in a single toolchain. This view is also used to visualize the last execution of the toolchain. In this case, real artifacts and trace information about tool executions are visualized. A tool capable of visualizing toolchain configurations is the *Validas Tool Chain Analyzer*[2]. Examples for tools visualizing toolchain traces are *ExplorViz*[3] and *ProM*[4]. As we suggest to use the same view to display both, configurations and traces, it must be hinted, which of both is actually shown.

Artifact Dependencies: From the toolchain view, the user is enabled to display the dependencies of any involved artifact, which is used as input, output, intermediate result or part of a tool's source code. Selecting an artifact results in a perspective switch to display only artifacts and dependencies between them as a directed dependency graph. Here, an approach similar to the *Massey Architecture Explorer*[5] can be used. This visualization tool displays static dependencies between Java classes and packages and could be easily adapted to display any kind of artifact dependencies of MDD projects.

Models: When selecting a model file from one of the perspectives, it is possible to represent it in its own notation. This notation can be textual [Sc12] or graphical [Ru16]. To display dependencies between models, the graph visualization described before could be used. In addition, a filter may be required to focus on models only for this use case.

Tool Artifacts: When selecting a tool from the toolchain view, the perspective changes to the tool view. Here, the generator's artifacts, namely templates and Java artifacts, are shown. Dependencies between those artifacts can also be visualized by the directed dependency graph, but there exists another tool specific perspective for this. In *ExplorViz* exists a possibility to view relations between artifacts of an application in a city like visualization. Real communication is visualized by this tool and there is a possibility to "zoom into" a single tool from a tool landscape perspective. Again, we propose to use this perspective for both, configuration and trace information.

Hierarchy: As we assume that artifacts are organized in a file system, a file explorer view is displayed in parallel to other perspectives. This view supports expanding artifacts such that elements defined within artifacts (e.g. model elements within model files) can also be displayed. For Java artifacts this is supported by the package explorer of the *eclipse*

[2] http://www.validas.de/TCA.html
[3] https://www.explorviz.net/
[4] http://www.promtools.org/
[5] http://xplrarc.massey.ac.nz/

IDE[6] for example. Moreover, *Sonargraph*[7] can display references in addition to contains-relations in a similar view.
Artifact-based Architecture: In MDD projects architectures can be defined on various levels as discussed in section 2. For each of these levels, different analysis tools like *HUSACCT*[8] or *ARAMIS* [Ni15] can be adapted to display artifact-based architectures. Such a perspective is added to the overall approach to display dependencies between architectural structures, which increase the understanding of MDD projects on a more abstract level.
Analysis Data: We assume that specific analyses of the overall data visualized so far are needed and that their results are visualized, too. Metrics are a special form of analysis, whose results are graphically displayed. Existing visualization techniques such as those used in *Sonarqube*[9] can be used. Other analyses result in a subset of dependencies and elements, which can be visualized by reusing the respective part of the integrated approach.

4 Conclusion

In this paper, we motivated the need for visualization tools supporting MDD projects and project analysis. Moreover, we presented our view on MDD projects focussing on tools and artifacts in the development phase of the project. Based on this, our vision of an integrated visualization approach was sketched taking into account existing approaches applicable for different development scenarios. As outlook to this work, we want to (1) ease the focus we put on MDD projects by taking into account the overall development process, (2) compare project data at different points in time, and (3) take into account more complex scenarios in which a tool generate parts of another tool used in the MDD project.

References

[BS03] Bederson, Benjamin B; Shneiderman, Ben: The craft of information visualization: readings and reflections. Morgan Kaufmann, 2003.

[CH03] Czarnecki, Krzysztof; Helsen, Simon: Classification of model transformation approaches. In: Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture. volume 45. USA, pp. 1–17, 2003.

[Gr08] Grönniger, Hans; Krahn, Holger; Rumpe, Bernhard; Schindler, Martin; Völkel, Steven: MontiCore: a framework for the development of textual domain specific languages. In: 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008, Companion Volume. pp. 925–926, 2008.

[Ni15] Nicolaescu, Ana; Lichter, Horst; Göringer, Artjom; Alexander, Peter; Le, Dung: The aramis workbench for monitoring, analysis and visualization of architectures based on run-time interactions. In: Proceedings of the 2015 European Conference on Software Architecture Workshops. ACM, p. 57, 2015.

[Ru16] Rumpe, Bernhard: Modeling with UML: Language, Concepts, Methods. Springer, 2016.

[Sc12] Schindler, Martin: Eine Werkzeuginfrastruktur zur agilen Entwicklung mit der UML/P. Aachener Informatik-Berichte, Software Engineering, Band 11. Shaker Verlag, 2012.

[6] http://www.eclipse.org/
[7] https://www.hello2morrow.com/products/sonargraph
[8] http://husacct.github.io/HUSACCT/
[9] http://www.sonarqube.org/