[BGRW18] A. Butting, T. Greifenberg, B. Rumpe, A. Wortmann:
On the Need for Artifact Models in Model-Driven Systems Engineering Projects.
In: Software Technologies: Applications and Foundations, LNCS 10748, Springer, 2018.
www.se-rwth.de/publications

On the Need for Artifact Models in Model-Driven Systems Engineering Projects

Arvid Butting, Timo Greifenberg(✉), Bernhard Rumpe, and Andreas Wortmann

Software Engineering, RWTH Aachen University, Aachen, Germany
greifenberg@se-rwth.de
http://www.se-rwth.de

Abstract. Model-driven development has shown to facilitate systems engineering. It employs automated transformation of heterogeneous models into source code artifacts for software products, their testing, and deployment. To this effect, model-driven processes comprise several activities, including parsing, model checking, generating, compiling, testing, and packaging. During this, a multitude of artifacts of different kinds are involved that are related to each other in various ways. The complexity and number of these relations aggravates development, maintenance, and evolution of model-driven systems engineering (MDSE). For future MDSE challenges, such as the development of collaborative cyber-physical systems for automated driving or Industry 4.0, the understanding of these relations must scale with the participating domains, stakeholders, and modeling techniques. We motivate the need for understanding these relations between artifacts of MDSE processes, sketch a vision of formalizing these using artifact models, and present challenges towards it.

1 Motivation

The complexity of future interconnected cyber-physical systems, such as the smart future, fleets of automated cars, or smart grids poses grand challenges to software engineering. These challenges partly arise from the number of domains, stakeholders, and modeling techniques required to successfully deploy such systems. Model-driven engineering has shown to alleviate this, but introduces the challenge of managing the multitude of different artifacts, such as configuration, models, templates, transformations, and their relations as contributed by the experts of different domains. Considering, for instance, software engineering for the factory of the future [10], successful deployment of a virtual factory [9] requires integration of modeling techniques to describe the factory's geometry, production processes and their optimization, software architecture, production systems with their interaction, manufacturing knowledge, and, ultimately,

This research has partly received funding from the German Federal Ministry for Education and Research under grant no. 01IS16043P. The responsibility for the content of this publication is with the authors.

M. Seidl and S. Zschaler (Eds.): STAF 2017 Workshops, LNCS 10748, pp. 146–153, 2018.
https://doi.org/10.1007/978-3-319-74730-9_12

general-purpose programming language artifacts. The artifacts contributed by the respective domain experts are required in different stages of the development process and exhibit various forms of relations, such as design temporal dependencies, run-time dependencies, or creational dependencies (*e.g.,* a model and the code generated from it).

Moreover, how artifacts interrelate not only depends on their nature, but also on the context they are used in and the tools they are used with. For instance, software architecture models may be used for communication and documentation, model checking, transformation into source code, or simulation of the system part under development. In these contexts, the relations required to understand and process such an artifact may change: whereas the pure architecture model might be sufficient for communicating its structural properties, transformation into source code relates it to transformation artifacts and to the artifacts produced by this transformation.

This paper extends previous work [5] in greater detail for a better understanding on how explicating these artifacts and their relations facilitates traceability of artifacts, change impact analysis [1], and interoperability of software tools all of which are crucial to successful model-driven engineering of the future's systems of systems.

2 Modeling Artifact Relations

Typical MDSE projects require a multitude of different artifacts addressing the different domains' concerns (cf. Fig. 1). Managing the complexity of these artifacts requires understanding their relations, which entails understanding the relations between their languages as well as between the development tools producing and processing artifacts. We envision a MDSE future in which these relations are made explicit and machine-processable using modeling techniques. To this end, we desire reifying this information as first-level modeling concern in form of an explicit *artifact model* defined by the lead architect of the overall MDSE project. Such a model precisely specifies the kinds of artifacts, tools, languages, and relations present in the MDSE project an thus enables representing the MDSE project in a structured way.

Such an artifact model should be capable to describe all different situations in terms of present artifacts and relations that could arise during its lifetime. The current situation of the project can be inspected by automatically extracting artifact data from the project according to the artifact models' entities and relations. This data corresponds to the artifact model ontologically, *i.e.,* represents an instance of it at a specific point in time. Analysts or specific software tools can employ this data to produce an overview of the current state, reporting issues, and identifying optimization potentials. Ultimately, this aims at enabling a more efficient development process.

To this end, the artifact model comprises, among others, the organization of artifacts in the file system, the configuration of the tool chain, the trace of the last tool chain execution as well as static knowledge relations between artifacts

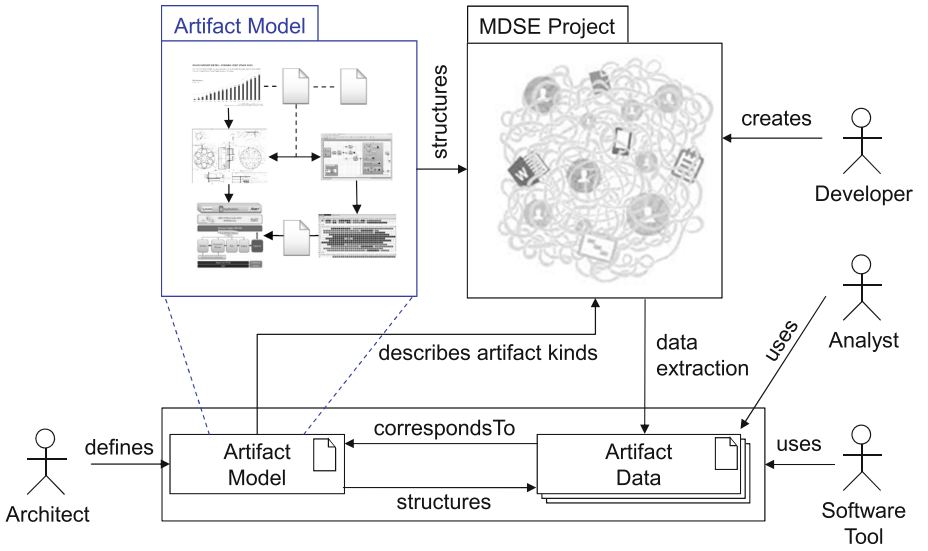

Fig. 1. An artifact model structures the different kinds of artifacts within MDSE projects. Corresponding artifact data enables analyses of the project state by analysts and software tools.

leading to an architectural view including input models, artifacts forming specific tools or the target product, artifacts managed by the tools, output artifacts, and handcrafted artifacts.

This model depends on the technologies and tools used to develop the target product. Hence, it must be tailored specifically to each MDSE project. Globally, parts of such a model could be reused from similar projects (which might be achieved employing language engineering and composition methods on the artifact modeling language). For instance, model parts describing the interfaces of tools could be reusable as well as the types of specific artifacts and their relations might be applicable to multiple projects. Nevertheless, we assume each project will require manual artifact modeling to adjust existing structures. Ultimately, creating such an artifact model would

- ease communication, specification, and documentation of artifact, tool, and language dependencies,
- enable automated dependency analysis between artifacts and tools,
- support change impact analysis in terms of artifact tool, or language changes,
- support checking compliance of tools and proposing artifact, tool, and relation adaptations to 'glue' tool chains,
- facilitate an integrated view on the usage of tools in concrete scenarios,
- enable data-driven decision making, and
- enable computation of metrics and project reports to reveal optimization potentials within the tool chain and the overall MDSE process.

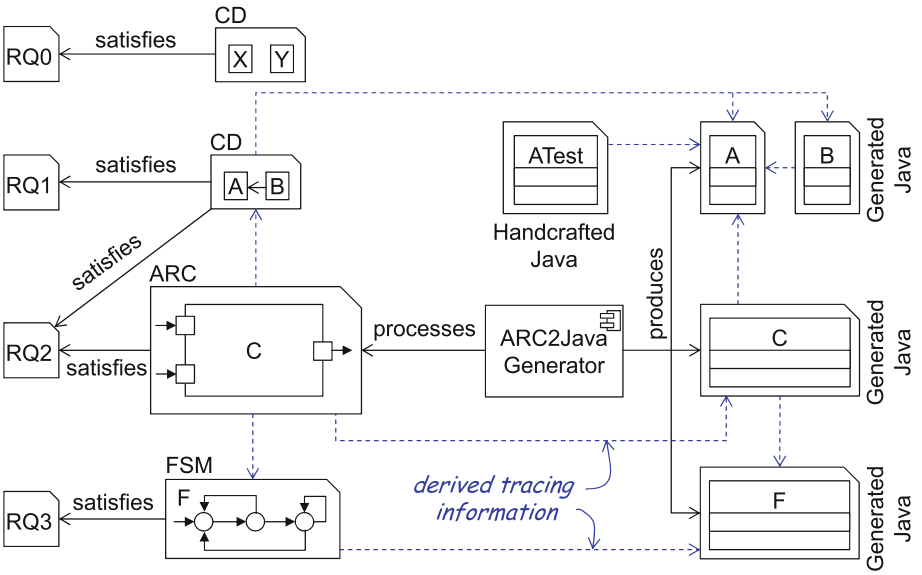

Fig. 2. Overview of explicit and implicit relationships between elements in an MDSE process, where associations colored black are explicitly specified and dashed, blue associations are implicitly defined within the underlying languages of the respective artifact, or derived by taking into account creational dependencies across different stages of a process. (Color figure online)

3 Example

Consider a company developing a software system using MDSE methodology in a multi-stage process. A common challenge in MDSE processes is to trace the impact of changes within an artifact to related artifacts across multiple stages in the process, and to detect implicit dependencies of different artifacts. An excerpt of artifacts used in an exemplary process in context of a single employed tool is depicted in Fig. 2. First, the company specifies requirements (*cf.* RQ0, ..., RQ3 in Fig. 2) that the functionality of the developed software should satisfy. In the next phase, models that reflect the specified requirements are implemented by modeling experts of the company. Each requirement defines assumptions that are satisfied by one or more modeling artifacts. The modeling artifacts conform to different modeling languages such as, *e.g.*, class diagrams, an architecture description language, and a finite state machine language as depicted in Fig. 2.

The connections between requirements and the modeling artifacts are defined manually. Then, the `ARC2JavaGenerator` tool is employed to transform the given heterogeneous models into Java code. A common difficulty, especially in large software projects, is to understand the exact mapping between each input and output artifact(s) of a tool execution. This mapping is usually encapsulated within the tool. Further, modeling languages typically allow several different

kinds of relationships between models and understanding these requires knowledge about the languages.

With an explicit artifact model as envisioned, the relationships between models and generated Java classes can be derived. Extracting derived information, from various sources, such as, *e.g.,* import statements in models, as well as the relationships between generated artifacts and the input models, greatly supports developers in analyzing dependencies in MDSE projects. For instance, the relations derived in Fig. 2 (denoted by dashed, blue arrows) could facilitate tracing and impact analysis as follows:

(1) To evaluate, which generated artifacts are needed to satisfy a specific requirements, this information can be derived taking into account the relations between models and requirements, and the relations between a model and the generated artifacts.
(2) To evaluate, which test case verify which of the requirements, the method in (1) can be used and, in addition, the relations between test and generated code must be regarded.
(3) To evaluate whether there exist unused model files, which lead to unnecessary complexity for the MDSE project. Those models can be identified by the mapping between input and output artifacts. If there is no mapping for an input artifact, this artifact is a candidate for removal.
(4) To evaluate whether there exist unused source code files. In our example, the generated Java class `B` is unused, as there is no derived association to this class from the Java class `C` generated from the architecture model, which is the excerpt under investigation here. Identifying unused files that do not need to be tested, packaged, or deployed leads to a more efficient MDSE process.
(5) To evaluate whether the transformation satisfies all requirements, the method in (1) can be used to determine all satisfied requirements. In the example, the class diagram containing the classes `X` and `Y` are not referenced from the architecture model, but the class diagram satisfies RQ0 and has no relation to parts of the generated code. This may indicate erroneous models.
(6) To estimate costs for modifications, tracing can calculate, which artifacts are affected by the modification of a certain artifact. The granularity of the distinct types of relations investigated in an artifact model influences the quality of such a change impact analysis.

4 State of the Art

There are various approaches to support model-driven systems engineering. Popular system engineering tools, such as *PTC Integrity Modeler*[1], *Syndeia*[2], or *Cameo Systems Modeler*[3] support modeling with SysML [8] and, hence, are able

[1] https://www.ptc.com/en/model-based-systems-engineering/integrity-modeler.
[2] http://intercax.com/products/syndeia/.
[3] https://www.nomagic.com/products/cameo-systems-modeler.

to manage different kinds of development artifacts in the same tool. One important feature of those tools is tracing between the managed artifacts. Nevertheless, these tools cannot be used out of the box for our intended purpose: (1) SysML is a general purpose modeling language, *i.e.,* it can be used to describe a large variety of systems. However, as domain-specific modeling advocates tailoring the modeling techniques to the participating domains, using general-purpose modeling languages usually leads to a coarse interpretation of diagrams with respect to the domains. This lack of precision, prevents leveraging the potential of fully automated, integrated model processing. Consequently, additional formalization is need to ensure that the semantics of modeled artifacts and relations between them is unambiguous. This, however, is either not supported by such tools at all or very limited (*cf.* stereotypes). (2) As we are especially interested in the coherences of MDSE processes, there is a need to inspect the artifacts, relations, and processes of MDSE tools themselves. Considering, for instance, highly customizable code generators, it rarely is fully understood which of the artifacts are in use in the context of a given project at a specific time. Especially when parts of MDSE tools are automatically generated themselves, tracing of the overall build process in terms of its artifacts becomes more challenging. This challenge can also not be solved by the mentioned tools, as they usually do not take the development tools into account in such a way, but focus on the development artifacts instead. In MDSE, proper model management is crucial when working with large collections of models. In [3] the notion of megamodels was introduced, which still subject to ongoing research [12,13]. Under the assumption that everything is model [2] one could argue that the artifact models proposed in this work are megamodels too. As we require formal encoding of models and their relations, there is a difference between megamodels and the proposed artifact models from our viewpoint. The elements of megamodels represent models and the links represent relationships between models [12]. The proposed artifact models focus on the model-driven build process including a whitebox view of the model-driven development tools. For instance, model elements of artifact models can also represent tools, artifacts the tools consists of, generated or handcrafted code files of the target system or configuration files. Links are then relationships between any of those elements. Nevertheless, there are commonalities with megamodels as models and their relationships can also become an important part in artifact models and the corresponding project data.

5 Challenges of Artifact Modeling

There are few approaches towards such an artifact model. The author of [4] focus on the integration of tools and the specification of tool chains and transformations between artifacts. Thus, artifacts managed within different tools are related to each other. The authors of [11] focus on an artifact-oriented way to describe a model-based requirements engineering process. Both approaches consider the requirement and design phases of MDSE projects only, but do not take code generation phases or implementation phases into account. Also, the tools

themselves are not considered in the presented models. The authors of [6], contributed the idea of providing project data to analysts and software tools, but do not combine this idea with an explicit artifact model. Hence, there are still open challenges, which have to be overcome towards efficient and sophisticated artifact modeling.

Methodology. The definition of a methodology on how to create artifact models tailored to the needs of a particular MDSE project. This includes:

- defining the scope of the MDSE project where artifact modeling can help taming the complexity,
- the development and selection of suitable modeling languages, tools and guidelines,
- the creation of model libraries providing reusable concepts common for system engineering projects, and
- development of reusable algorithms based on artifact models providing valuable analysis for common problems of system engineering projects.

Tools. Defining mechanisms, tools, and infrastructure supporting extraction and understanding of artifact data, including

- visualization capabilities, such as those proposed in [7],
- a methodology for integrating the different automated analysis tools to a given infrastructure,
- common interfaces for accessing artifact data, and the
- handling large amounts of artifact data efficiently.

Integration. Overcome modeling challenges, such as

- providing ways of defining and ensuring compliance between related software tools, such as editors, generators, or transformations, and
- integrating process data and historical data into such an artifact model to enable comprehending the state and changes of artifacts and their relations over time.

6 Conclusion

Model-driven development facilitates systems engineering. However, to this end it introduces new challenges, out of which taming the complexity of participating artifacts and their relations is a very important one. We argue that investigating and reifying these using an artifact model and corresponding tooling is crucial to the successful deployment of future systems of systems. The ultimate goal would be, that architects can model their project, including the tools, the MDSE process and the target system's architecture with all relevant relations with minimal effort. The corresponding data should be extracted automatically and enable overviewing of the project's current state. This enables making data-driven decisions regarding tool landscape, processes, and architectures such that

the future MDSE projects can be run successfully. In this paper we presented a small example clarifying the problem. Nevertheless, in complex scenarios with multi-level generation processes and where models of different engineering discipline are related to describe the target product, the benefit of our approach will unfold to full extend. To guide this, particular challenges of artifact modeling future research should investigate were highlighted.

References

1. Arnold, R.S.: Software Change Impact Analysis. IEEE Computer Society Press, Los Alamitos (1996)
2. Bézivin, J.: On the unification power of models. Softw. Syst. Model. **4**(2), 171–188 (2005)
3. Bézivin, J., Jouault, F., Valduriez, P.: On the need for megamodels. In: Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development Workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (2004)
4. Braun, P.: Metamodellbasierte Kopplung von Werkzeugen in der Softwareentwicklung. Logos, Berlin (2004)
5. Butting, A., Greifenberg, T., Rumpe, B., Wortmann, A.: Taming the complexity of model-driven systems engineering projects. In: Cabot, J., Paige, R., Pierantonio, A. (eds.) Part of the Grand Challenges in Modeling (GRAND 2017) Workshop (2017). http://www.edusymp.org/Grand2017/
6. Czerwonka, J., Nagappan, N., Schulte, W., Murphy, B.: CODEMINE: building a software development data analytics platform at Microsoft. IEEE Softw. **30**(4), 64–71 (2013)
7. Greifenberg, T., Look, M., Rumpe, B.: Visualizing MDD projects. In: Software Engineering Conference (SE 2017). LNI, pp. 101–104. Bonner Köllen Verlag (2017)
8. Object Management Group: OMG Systems Modeling Language (OMG SysML), May 2017. http://www.omg.org/spec/SysML/1.5/
9. Jain, S., Lechevalier, D.: Standards based generation of a virtual factory model. In: Proceedings of the 2016 Winter Simulation Conference, pp. 2762–2773. IEEE Press (2016)
10. Khan, A., Turowski, K.: A survey of current challenges in manufacturing industry and preparation for industry 4.0. In: Abraham, A., Kovalev, S., Tarassov, V., Snášel, V. (eds.) Proceedings of the First International Scientific Conference "Intelligent Information Technologies for Industry" (IITI' 16). AISC, vol. 450, pp. 15–26. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33609-1_2
11. Méndez Fernández, D., Penzenstadler, B.: Artefact-based requirements engineering: the AMDiRE approach. Requir. Eng. **20**(4), 405–434 (2015)
12. Salay, R., Kokaly, S., Chechik, M., Maibaum, T.: Heterogeneous megamodel slicing for model evolution. In: ME@ MODELS, pp. 50–59 (2016)
13. Simmonds, J., Perovich, D., Bastarrica, M.C., Silvestre, L.: A megamodel for software process line modeling and evolution. In: 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 406–415. IEEE (2015)