Nahtlose Systementwicklung durch Werkzeugintegration über Komponenten- und Konnektorarchitekturen

Towards Sustainable Systems Engineering – Integrating Tools via Component and Connector Architectures

M. Dalibor, N. Jansen, J. Michael*,
B. Rumpe, A. Wortmann

Software Engineering, RWTH Aachen University
Ahornstraße 55, 52074 Aachen, Germany

[DJM+19] M. Dalibor, N. Jansen, J. Michael, B. Rumpe, A. Wortmann:
Towards Sustainable Systems Engineering-Integrating Tools via Component and Connector Architectures.
In: Antriebstechnisches Kolloquium 2019: Tagungsband zur Konferenz, pp. 121-133, Feb. 2019.
www.se-rwth.de/publications/

Inhaltsverzeichnis

Abstract

The engineering of sophisticated systems today is highly multi-disciplinary and depends on domain-specific documents being exchanged between the different participating experts, software tools, and engineering phases. The lack of integration between these tools raises the challenge of media disruption, which demands for manual translation of documents passed between the different tools. Existing research on tool integration focuses the pairwise translation between specific technological spaces only, which hinders toolchain extension. We conceived an integration model for systematic data exchange among systems engineering tools that leverages component and connector architectures. Our model supports automated data translation and distribution between tools encapsulated using the architecture description language MontiArc and model transformations. This enables agile and sustainable development among heterogeneous toolchains, which preserves existing workflows and is easily extensible.

1 Introduction

In modern development of mechatronic and cyber-physical systems, a variety of software tools, such as MagicDraw [SIL09], support developers in creating, maintaining, and updating their products. These tools have been proven to be essential for various domains such as automotive [BLO13], avionics [FEI12], and robotics [WIG17]. As technology gets more advanced, the complexity of the resulting products and hence the necessary tooling grows in terms of required functionality diversity. Unfortunately, the available tools support the developer concerning specific tasks only. Due to the heterogeneity of particular tools, they lack in interconnectivity and uniform information distribution. This emerges media disruptions, the manual and error-prone translation between tool data formats, which impedes agile development.

Different approaches in Model-Based Systems Engineering (MBSE), such as SysML, support product development by providing tools that enable working on a combined system model [ROB98]. While these solutions tackle the problem of distributed data by joining information into a single source of truth, they also entail additional effort. Domain experts have to be trained and motivated in using these new tools. Additionally, companies often require customizations that may not be compatible with the prescribed tooling [DRA18]. Developers also tend to fall back into accustomed workflows with familiar tools, if the added value of systems engineering is not accessible directly [CHA18].

These challenges require a domain-independent solution that prevents media disruptions, without entailing additional effort for domain experts. Therefore, we present our approach of leveraging component and connector (C&C) architectures to automatically distribute tool-specific data as generalized messages along a generated communication layer towards particular tools. Hence, developers can preserve their workflows within familiar tools, while automatically exchanging information throughout the toolchain. The

approach is based on the architecture description language (ADL) MontiArc [BUT17a, BUT17b, RIN15] for C&C systems and model transformations [HOE18].

In the following, Section 2 illustrates selected challenges in agile systems engineering and motivates connecting the different technological spaces of development tools. Section 3 presents preliminaries before Section 4 introduces the realization of our solution. Afterwards, Section 5 illustrates the benefits through a case study, and Section 6 considers related work. Finally, Section 7 discusses our solution and concludes.

2 Motivating Example

Consider a company producing wind turbines using highly heterogeneous software tools, such as MagicDraw [SIL09] for modeling software structure, CATIA [BRA09] for computer-aided design (CAD), MATLAB Simulink [ONG98] for simulation, and a spreadsheet software for production cost analysis. Domain experts have to consider multiple aspects when developing a new turbine design. For the sake of simplicity, we restrict this example to a small subset of the original parameter space. Figure 1 presents a class diagram for the simplified wind turbine infrastructure. The model of a wind turbine is characterized by its `height`, `span`, expected `efficiency`, and production `cost`. Additionally, it features `geometry` data that also contains `material` information.

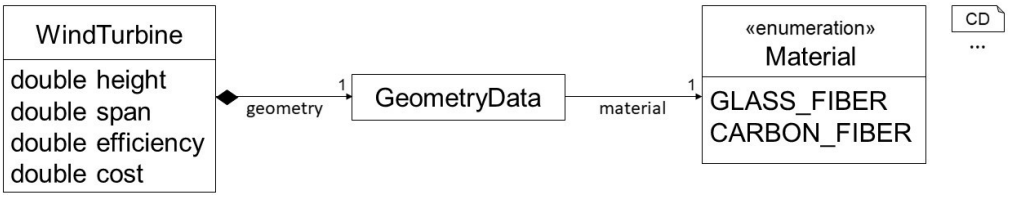

Figure 1: Class diagram of a simplified structure of a wind turbine. The wind turbine features `height`, `span`, `efficiency`, and production `cost`. Additionally, it requires `geometry data` with a corresponding `material`.

When designing a new wind turbine, developers have to address these parameters. A toolchain comprising a variety of tools supports the development process concerning specific focal aspects. In this example, we require four tools to properly model the wind turbine. In Figure 2, we illustrate the corresponding applications. Initially, we have a SysML Tool MagicDraw for defining the overall architecture. With this, constraints such as the `height`, rotor `span` width, and `efficiency` are specified. To evaluate these, we perform a MATLAB Simulink Simulation to verify over an according environment model, whether the wind turbine reaches the expected `efficiency` threshold. Additionally, we use the CAD Tool CATIA to specify `geometry data` including `material`. Finally, we perform a Cost Calculation to estimate production `cost`.

This toolchain is capable of completing the distinct tasks. However, the involved tools rely on information that is specified within another tool. Figure 2 presents an example where all these tools have data interdependency. The `CAD Tool` and the `Simulation`

both require constraints depicted in the `SysML Tool`. Additionally, the `Cost Calculation` depends on the `SysML Tool` (for `height` and `span`), as well as the `CAD Tool` (for the `rotor material`). Since these tools are not connected, development entails media disruptions through manual translation of output data of one tool into input data of another tool, which impede severely development. Moreover, this constitutes an unnecessary source of failure and prevents truly agile systems engineering, since the data has to be traced down the complete toolchain, once a parameter changes. To mitigate this, the next sections emphasize a method to automatically distribute the data throughout the toolchain without media disruptions.

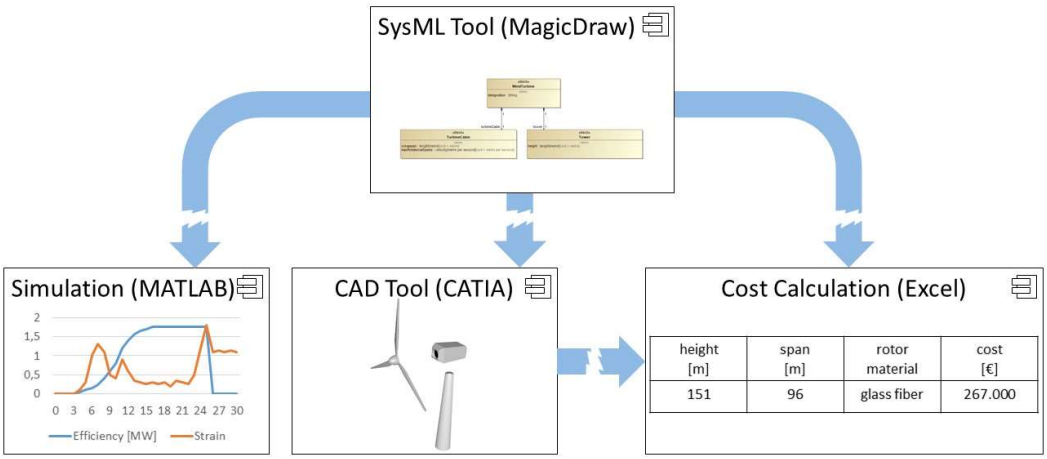

Figure 2: Exemplary tooling landscape for the development of a wind turbine. Media disruptions emerge between the four distinct tools. Information is not distributed throughout the system automatically.

3 Preliminaries

The realization of connecting heterogeneous tools relies on the MontiArc ADL and model transformations across the different technological spaces. This section introduces both.

MontiArc [BUT17a, BUT17b] is an ADL for modeling C&C systems. An architecture designed with MontiArc consists of components, ports, and connectors that define the C&C infrastructure over a corresponding configuration. Components either compute behavior or contain configurations of subcomponents. They feature interfaces of typed, directed ports to exchange messages. We will leverage the strongly typed component interfaces of components to encapsulate tool-specific inputs and outputs and use MontiArc's composition mechanism to integrate different encapsulated tools without media disruptions.

Within the components we employ formal model-to-model (M2M) transformations [HOE18] to automatically translate messages received by the tool-encapsulating components into suitable inputs for the contained tool. Approaching tool integration via encapsulation and model transformation enables portability between the different data formats (or metamodels [KLE08]) and facilitates building bridges among the different technological spaces of systems engineering.

We apply model transformations on the abstract syntax tree (AST), a tree representation of the parsed model that only contains processable features, without further syntactic sugar, such as file format artifacts. The overall structure of the abstract syntax is given by the metamodel [KLE08], the formal definition of a language or data format. This metamodel prescribes the set of possible models within the corresponding domain.

4 Constructing a Tool Communication Infrastructure with C&C Architectures

In this section, we explain our approach of enabling an automatic data transfer between distinct tools, utilizing MontiArc C&C architectures. The basic idea is to create a transportation layer by using connected components that exchange messages. Tools can be plugged in into components, which serve as communication interfaces and distribute required data along the network. In the following, we explain, how components extract the required information from an application concerning its metamodel and transform it into a generalized form. Figure 3 illustrates the structure of such a tool interface. A component contains multiple ports, one for each data set that should be distributed. We use an incoming port when the tools require information that is provided elsewhere. Analogously, outgoing ports denote that the tool contributes information for other applications. The interdependencies of the tools are lifted onto the transportation layer of the C&C infrastructure. Here, they are realized via corresponding connectors between ports of the particular components. This abstraction enables data exchange with respect to but also independent of the actual interdependencies of the underlying tooling landscape.

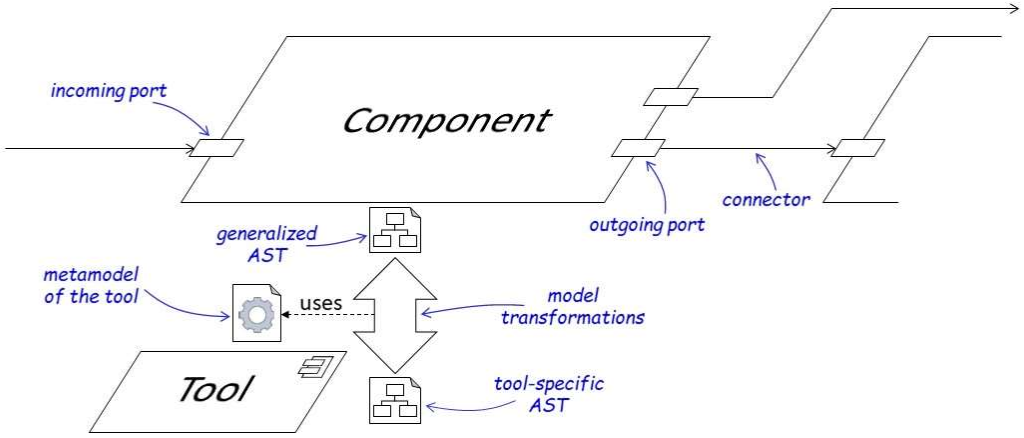

Figure 3: Components serve as communication interfaces to the C&C transportation layer for distributing data to other tools. Model transformations enable the translation of tool-specific data into a generalized and distributable format.

Since we can realize the communication on the C&C layer, we now consider the transportation of information from a tool towards its corresponding component. To this end, we use model transformations to prepare the data for exchange. Figure 3 sketches the translation of tool-specific data into a generalized model for data distribution. The data created within a tool gets parsed with respect to the corresponding metamodel of the

tool. The information is stored in an AST, and we apply M2M transformations to map the tool-specific abstract syntax representation to a generalized form, which is used on our transportation layer. Hence, the infrastructure extracts data from a tool and distributes it along the network.

To provide information for a tool, we use an analogous approach. Here, we translate the generalized AST into tool-comprehensive data. In general, there are two approaches. If the tool supports an application programming interface (API), we again use M2M transformations to map the general AST into a specific that the tool can directly process. Otherwise, we generate a file with respect to the tool's metamodel. Thus, the tool can import the required information.

This enables a complete data distribution throughout a tooling landscape of highly heterogeneous applications. Data is exchanged via model transformations between tools and the corresponding components, which then distribute messages for communication along the C&C network. Finally, the data is reintegrated into the desired tools.

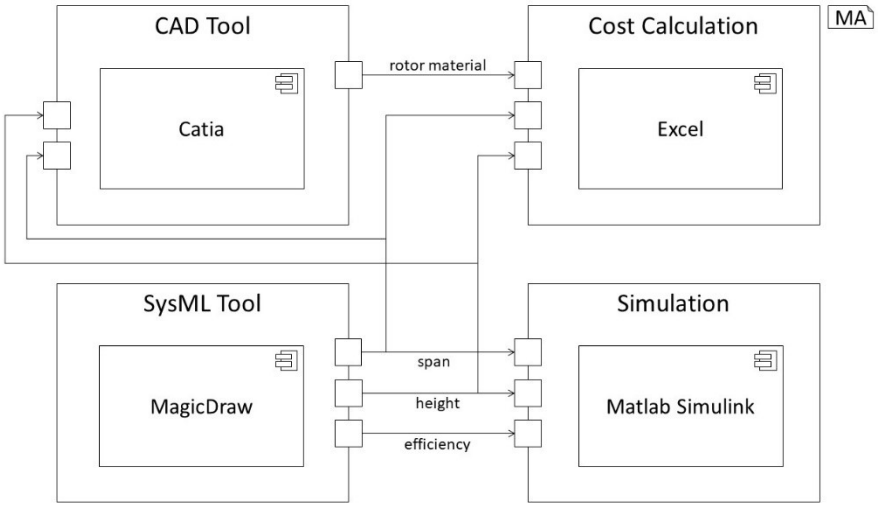

Figure 4: C&C system as communication layer for distributing data among distinct tools. Components exchange information with their underlying tools and transport messages concerning the original dependencies.

5 Case Study

To evaluate our approach of generating a communication infrastructure with C&C architectures, we consider the initial example from Section 2. Again, we require four distinct applications for designing a wind turbine. For each tool, we create a component in the communication infrastructure to distribute and receive data. Figure 4 illustrates the MontiArc C&C model for the tooling landscape. Each tool is encapsulated by its corresponding component that handles the data distribution. The ports and connectors are derived from the original tool interdependencies described in Section 2. The `SysML Tool` provides the `span` and `height` values for the `Simulation`, `CAD Tool`, and `Cost Cal-`

culation. Additionally, it passes the expected efficiency to the Simulation. Finally, the CAD Tool contributes the rotor material that is required for the Cost Calculation as well. Thus, the C&C infrastructure distributes information across the communication network. The particular components translate received messages into consilient tool-specific information. This way, the communication layer ensures global consistency, even among highly heterogeneous tools and thus, enables agile MBSE.

6 Related Work

There have been many attempts to integrate tools that support the MBSE process [FER15, SHA10, CHA06]. The realized solutions are often particular for tools that companies use to define their system models. For example, in [FER15] the authors present an integration between CAD systems tailored for shipbuilding and general product lifecycle management (PLM) tools. The key idea is the publish and synchronization mechanism which ensures that changes in one tool are transferred to the other. Publishing refers to the transfer of changes in the CAD tool into an PLM. Synchronization, on the other hand, is the process of informing the CAD system about changes in the PLM system. The whole process of publishing and synchronizing relies on a database that contains the identification between CAD and PLM elements. In contrast to our approach, this approach is specific for CAD systems, whereas we present a concept that integrates arbitrary tools. Besides, we provide a graphical notation of tool interactions, which improves readability compared to a database table.

Another approach introduces an integration idea based on a common SysML model, that describes those parts of the system that are common for all tools that interact during the systems modeling process [SHA10]. A mapping between the SysML model and the tool-specific models is archived by model transformations, that define which parts of the SysML model are relevant for the specific tool and how they are represented. The paper evaluates the idea based on an integration between EPLAN [SEN09] and Modelica [FRI10]. The main advantage of C&C models is the visualization. It shows explicitly which data is transferred between involved tools. This enables engineers to analyze how changes in one tool may affect models that engineers created and maintain in another tool. Therefore, as effects of a change in one model are also visible, simplifies an agile systems engineering.

7 Discussion and Conclusion

We presented a concept of connecting heterogeneous tools by leveraging C&C architectures and model transformations. The goal was preventing media disruptions in development due to tool incompatibility. At its core, components provide communication interfaces for the underlying applications. We presented the generalization of tool-specific data using M2M transformations concerning an underlying metamodel. The port and connector configuration, which results from the original data dependencies, represents

a communication layer. This allows the exchange of messages that finally, are translated back into tool-specific information. As communication is automated, this approach does not entail any additional effort for domain experts. In fact, our solution reduces the amount of work, as manual data distribution is now managed by the system. This also eliminates the entailed source of error and facilitates agile development. In contrast to existing solutions that mainly concentrate on data transfer between two particular tools, our approach describes a general concept independent of underlying applications. Since the established communication infrastructure is concealed from the development environment of the domain experts, our solution perfectly scales even for larger systems. We believe that connecting tools among distinct technological spaces is essential for future development of mechatronic and cyber-physical systems.

8 References

[RIN15] Ringert, Jan Oliver and Roth Alexander and Rumpe, Bernhard and Wortmann Andreas: Language and Code Generator Composition for Model-Driven Engineering of Robotics Component & Connector Systems. In: Journal of Software Engineering for Robotics.

[BUT17a] Butting, Arvid and Kautz, Oliver and Rumpe, Bernhard and Wortmann, Andreas: Architectural Programming with MontiArcAutomaton. In: 12th International Conference on Software Engineering Advances (ICSEA 2017).

[BUT17b] Butting, Arvid and Haber, Arne and Hermerschmidt, Lars and Kautz, Oliver and Rumpe, Bernhard and Wortmann, Andreas: Systematic Language Extension Mechanisms for the MontiArc Architecture Description Language. In: European Conference on Modelling Foundations and Applications, 2017.

[BLO13] Blom, Hans et al.: EAST-ADL: An architecture description language for Automotive Software-Intensive Systems. In: Embedded Computing Systems: Applications, Optimization, and Advanced Design: Applications, Optimization, and Advanced Design, 456, 2013.

[FEI12] Feiler, Peter H. and Gluch, David P: Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language, Addison-Wesley, 2012.

[WIG17] Wigand, Dennis Leroy and Nordmann, Arne and Dehio, Niels and Mistry, Michael and Wrede, Sebastian: Domain-Specific Language Modularization Scheme Applied to a Multi-Arm Robotics Use-Case. In: Journal of Software Engineering for Robotics, 2017.

[DRA18] Drave, Imke and Greifenberg, Timo and Hillemacher, Steffen and Kriebel, Stefan and Kusmenko, Evgeny and Markthaler, Matthias and Orth, Philipp and Salman, Karin Samira and Richenhagen, Johannes and Rumpe, Bernhard and Schulze, Christoph and von Wenckstern, Michael and Wortmann,

Andreas: SMArDT modeling for automotive software testing. In: Software: Practice and Experience, 1-28, 2018.

[CHA18] Chami, Mohammad and Bruel, Jean-Michel: A Survey on MBSE Adoption Challenges. In: INCOSE EMEA Sector Systems Engineering Conference, 2018.

[HOE18] Hölldobler, Katrin and Rumpe, Bernhard and Wortmann, Andreas: Software Language Engineering in the Large: Towards Composing and Deriving Languages. In: Computer Languages, Systems & Structures, 54:386-405, Elsevier, 2018.

[KLE08] Kleppe, Anneke: Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Pearson Education, 2008.

[SIL09] Šilingas, Darius and Butleris, Rimanta: Towards Implementing a Framework for Modeling Software Requirements in MagicDraw UML. In: Information Technology and Control, 38.2, 2009.

[ONG98] Ong, Chee-Mun: Dynamic Simulation of Electric Machinery: using MATLAB/SIMULINK. Vol. 5, Upper Saddle River, NJ: Prentice hall PTR, 1998.

[BRA09] Braß, Egbert: Konstruieren mit CATIA V5: Methodik der parametrisch-assoziativen Flächenmodellierung. Hanser Verlag, 2009.

[FER15] Fernández, Rodrigo Pérez and Lado, Roberto Penas: Integration between shipbuilding CAD systems and a generic PLM tool in naval projects. In: Computer Science, 2.5, 181-191, 2015.

[SHA10] Shah, Aditya A. et al.: Multi-view modeling to support embedded systems engineering in SysML. In: Graph transformations and model-driven engineering, 2010.

[CHA06] Chang, Kuang-Hua and Joo, Sung-Hwan: Design parameterization and tool integration for CAD-based mechanism optimization. In: Advances in Engineering Software, 2006.

[ROB98] Robertson, Tim: INCOSE Systems Engineering Handbook. INSIGHT, 1.2 Wiley Online Library, 1998.

[SEN09] Sendler, Ulrich: EPLAN Software & Service. In: Das PLM-Kompendium 163-174. Springer, Berlin, Heidelberg, 2009.

[FRI10] Fritzson, Peter: Principles of object-oriented modeling and simulation with Modelica 2.1. John Wiley & Sons, 2010.