[KRW+23] S. Koch, F. Reiche, S. Weber, M. Konersmann, S. Stüber,
L. Wollenhaupt, B. Taghavi, B. Rumpe, R. Heinrich:
A Collection of Scenarios for the Decomposition and Composition of Model-based Analys
Karlsruher Institut für Technologie (KIT), 2, Technical Report, Volume 2023(2),
Karlsruhe Reports in Informatics, Aug. 2023.

Karlsruhe Institute of Technology

Karlsruhe Reports in Informatics 2023,2
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

A Collection of Scenarios for the Decomposition and Composition of Model-based Analyses

Sandro Koch, Frederik Reiche, Sebastian Weber, Marco Konersmann, Sebastian Stüber, Lucas Wollenhaupt, Bahareh Taghavi, Bernhard Rumpe, and Robert Heinrich

2023

KIT – The Research University in the Helmholtz Association

A Collection of Scenarios for the Decomposition and Composition of Model-based Analyses*

Sandro Koch[1], Frederik Reiche[1],
Sebastian Weber[2], Marco Konersmann[3], Sebastian Stüber[3], Lucas Wollenhaupt[3], Bahareh Taghavi[1], Bernhard Rumpe[3], and Robert Heinrich[1]

[1] KASTEL – Institute of Information Security and Dependability
Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany
{sandro.koch|frederik.reiche|bahareh.taghavi|robert.heinrich}@kit.edu
[2] FZI Forschungszentrum Informatik
Karlsruhe, Germany
{sebastian.weber}@fzi.de
[3] Software Engineering Department of Computer Science
RWTH Aachen University, Aachen, Germany
{konersmann|stueber|wollenhaupt|rumpe}@se-rwth.de

Abstract. This technical report provides a collection of scenarios for the decomposition and composition of (modular) model-based analyses for different quality properties and different domains to serve as a basis for illustrative examples and evaluation scenarios in the FeCoMASS research project.

Keywords: model-driven engineering; model-based analysis; composition; decomposition; scenarios

1 Introduction

Before realising a complex software-intensive system it is worth analysing its emerging properties. Analyses are applied for investigating systems for quality properties or properties specific to a given domain. Model-based analysis is the appropriate technique to do that early in development to avoid inappropriate design decisions and costly design flaws. For today's heterogeneous and complex systems, analysis techniques become complex as well. To master the development of tailor-made analysis techniques, decomposition and composition of analyses is unavoidable. The very different nature of quality properties has led to the use of individual analysis techniques and independent tools for each quality property. Moreover, recently emerging innovations like internet of things and cyber-physical systems combine several domains, such as software, electrics/electronics and

* This work was funded by the DFG (German Research Foundation) – project number 499241390 (FeCoMASS).

mechanics. In consequence, analyses need to be (de)composed along quality properties and domain-specific properties.

The goal of the FeCoMASS research project is to provide more flexibility in model-driven engineering by investigating foundations of decomposition and composition mechanisms specifically for model-based analyses for tomorrow's increasingly heterogeneous and complex systems.

In this technical report, we describe scenarios for (de)composing (modular) model-based analyses for different quality properties and different domains to serve as a basis for illustrative examples and evaluation scenarios in the FeCoMASS project. For detailing and identifying the scenarios we will build upon knowledge and experience from (a) recent research initiatives (like the DFG Exzellenzcluster "Internet of Production" and the topic Engineering Secure Systems of the Helmholtz Association (HGF)), (b) scenarios identified during the Dagstuhl seminar 19481 [12,19], and (c) the evolution of historically-grown modelling languages and analysis approaches. For the historically-grown Palladio approach [30], for example, composition scenarios have been mentioned in [37] and [20]. We further investigate historically-grown analysis approaches in this report by surveying documentation and artifacts like code, models and language fragments in online repositories to define appropriate composition scenarios.

The report is structured as follows. In section 2, we first present foundations on the decomposition and composition of model-based analyses relevant for understanding the report. In section 3, we then present examples of historically-grown model-based analyses to serve as illustrative examples and evaluation scenarios for the decomposition and composition of model-based analyses in the FeCoMASS project. The report concludes in section 4 with a summary and outlook of future work in the project.

2 Foundations of Decomposition and Composition of Model-based Analyses

This section introduces fundamental concepts and their relationships necessary to understand the remainder of the report. An overview of these concepts and relationships is given in Figure 1 in form of an extended feature model.

A *feature model* [11] is a formalism to capture the variability and interdependencies of features of a specific subject. Based on a feature model, subsets of the given features are selected to specify which features are of current interest. In our previous work [23], we used feature models to specify interdependencies between language features and select those of current interest for language composition. In FeCoMASS, we will apply our notion of feature models to two dimensions: (a) modelling languages and (b) analysis techniques. Further, we will use feature models in FeCoMASS to specify the interdependencies between analysis features, language features and from analysis features to language features as well as to select those features (both analysis and language features) of current interest for analysis composition and analysis tool development. As shown in Figure 1,

parent-child relationships form a tree allowing, e. g., type mandatory, optional and alternative feature groups [11].

A *modelling language feature* is an abstraction of a thing to be modelled [23]. An *analysis feature* is an abstraction of a property to be analysed [18]. For sake of compositional reuse, it is of high interest to use analysis features, operating on language features. The concepts of analysis features are applied in FeCoMASS to decompose the increasing complexity of analysis techniques.

Language features are implemented by modelling language components. A *modelling language component* describes language constituents, e. g. through metamodels or grammars, has explicit interfaces and composition operators [9,23] for other modelling language components, and has an individual, composable semantics. Analysis features are implemented by *analysis component*s containing the analysis algorithms realised in source code. These analysis components are executable on the needed language features, have explicit interfaces and composition operators for other analysis components [18].

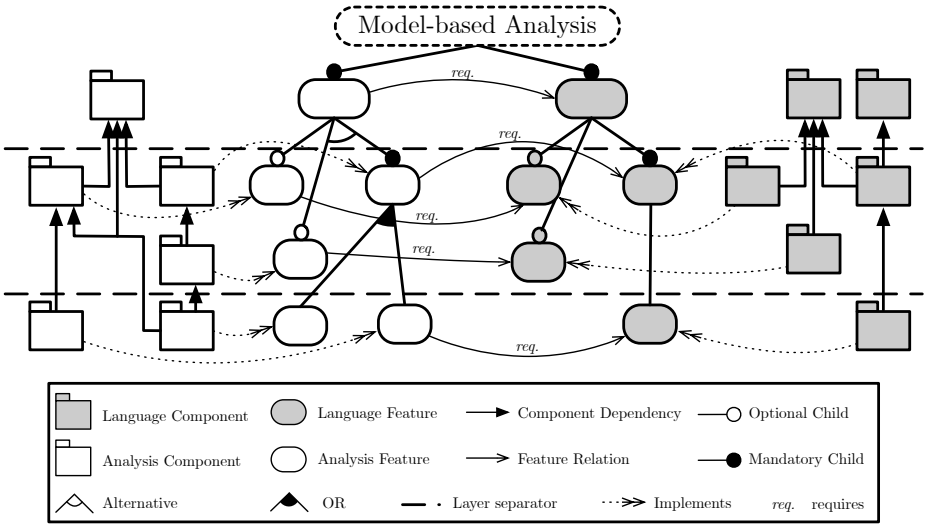

Fig. 1. Relationships between modelling language and analysis concepts: analysis features that are implemented by analysis components require (req.) modelling language features that are implemented by modelling language components.

A *model-based analysis* is a type of analysis that uses models for reasoning about the system and for communicating the results [42]. This type of analysis is formally described according to [39] in Equation 1 as a projection from models of an input modeling language M to a model of the output modeling language R. Therefore, we describe an analysis as a function $A(m) = r$ using an input model to provide an output model.

$$A\colon M \to R \tag{1}$$

A *modelling language* is created and applied to specify models to efficiently design and reason about systems [27]. Modellers can use standardised languages, such as UML [34] or SysML [14] or they can design their own domain-specific modelling languages [13,10]. To capture reoccurring domain knowledge, language workbenches [40,26] enable describing extensible languages. A modelling language consists of an explicit syntax and an associated semantics. The *syntax* describes the words and structure ("grammar") of the language. Denotational *semantics* is, e. g., realised by mathematically sound definition of a semantic mapping from well-formed models to an appropriate, well understood semantic domain [16]. Each modelling language usually has its own semantic domain. For example, statecharts use I/O-relations. Thus, the mathematical semantic construction to be used in a given case depends on the used modelling language and the scenario to be modelled. However, an integrated semantics typically needs a complex mathematical structure, capturing all relevant concepts of the systems under development. FOCUS [7,31] defines such a mathematical system model, where almost all other semantic domains and especially the SysML semantics can be embedded in. FOCUS furthermore provides a clear notion of refinement and composition [6]. Therefore, we use FOCUS as the conceptual, mathematical foundation for semantics in FeCoMASS, whenever the analyses cannot be based on a homogeneous smaller semantic domain anymore.

An *analysis technique* is applied for reasoning about structure, behaviour and/or quality of systems based on a model. Various different analysis techniques are possible, e. g., based on queuing networks or Markov chains. While we on the one hand belief in the hypothesis: *analysis techniques can (to a large extent) be decomposed into individual, reusable algorithms described in form of their "analysis features"*, we on the other hand investigate in detail the interactions between these analyses in FeCoMASS, because these analysis interaction points are the glue to combine analysis techniques to higher system understanding. An analysis technique is *semantically correct*, if the calculated analysis results conform to the mathematically defined semantics of the analysed models. Semantics is not necessarily computable, especially when behaviour is involved. Thus, a computable analysis technique may only be approximative, while a mathematical semantics definition usually is precise. The analysis technique is *valid in certain boundaries*, if the approximation deviation is small enough with respects to an appropriate metric. Therefore, numeric metrics, conservative estimations, and similar approximative forms of analysis results are of interest. A sound denotational semantics is therefore the basis to capture the correctness and validity of analysis techniques.

3 Decomposition and Composition Scenarios

In this section, we present examples of historically-grown model-based analyses to serve as illustrative examples and evaluation scenarios for the decomposition and composition of model-based analyses in FeCoMASS.

3.1 Palladio – Software Architecture-Based Quality Prediction

The Palladio approach [30] is an architecture-based performance prediction approach for software systems. It allows software architects to simulate and analyze the behaviour of their systems under different workloads and configurations, in order to identify bottlenecks and optimize performance. It is based on the Palladio Component Model (PCM), a domain-specific modeling language that allows modelling the structure and behaviour of software systems. Starting over 20 years ago, the PCM has a long history of evolution. In the time from spring 2007 to fall 2012, the PCM grew from under 100 to over 200 classes [38], and still continues to grow. There are at least 12 extensions of the PCM [23].

Not only the modeling language of the Palladio approach continuously grows, also the analysis techniques of the Palladio approach do. The Palladio approach comprises at least eight different analysis techniques for reasoning about software architectural design. Those are specific to certain quality properties (e.g., performance versus reliability), tools (e.g., the Palladio bench [30] versus QPN-Tool [3]) or analysis tasks (e.g., mean time analysis versus prediction of a statistical distribution).

The Palladio approach was initially designed for performance analysis based on software architectures. Over time, Palladio has been extended for modelling and analysing reliability [5], scalability and elasticity, maintainability [32], confidentiality [35], security [41], energy consumption [36] and many other quality properties [24]. These modifications led to serious degradation of the structure of the modelling language [37] and analysis techniques [20]. Feature overload, feature scattering and unconstrained creation of dependencies harm the evolvability and reusability of the modelling language and analysis techniques. This is because Palladio, as other similar approaches, relies on a monolithic modelling language and monolithic analysis techniques, making modifications and extension to new properties challenging. However, in each project a large set of these analysis techniques is not used, because they do not apply or have redundant alternatives.

SimuLizar: One of the performance analyses of the Palladio approach is SimuLizar [4]. SimuLizar comprises a set of analysis techniques that can analyze PCM instances. SimuLizar supports most of the PCM language features. As a historically-grown analysis technique (development started in 2013) SimuLizar shows the typical erosion of the internal structure over time and therefore is a good example of a historically grown analysis technique to apply the approaches developed in FeCoMASS. An detailed overview of design smells, such as feature scattering, global states and god parameters, in SimuLizar is given in [20].

SimuLizar currently comprises 75 packages, 306 classes, 69 interfaces, and three enumerations, organized into 36 Java projects. Over time, SimuLizar has undergone significant growth, with its size doubling since 2015, resulting in the number of classes increasing from approximately 150 to the current more than 300 classes. Most of these classes are contained in a big monolithic analysis component. Throughout its development, SimuLizar has experienced numerous evolutionary

changes. SimuLizar offers ten openly available extensions that resulted for example from research collaborations and student theses.

In FeCoMASS, we will develop approaches to decompose the monolithic structure of SimuLizar and thus make it more easier to understand, refactor and extend.

Furthermore, there are examples of model-based analyses that extend the Palladio approach by additional analysis features to form more comprehensive analysis techniques. These may serve as scenarios for the analysis composition approaches developed in FeCoMASS. Some of these examples are described in the following.

OMPCM: The OMNeT++ Palladio Component Model (OMPCM) [25] is a model-based analysis that leverages the capabilities of the OMNeT++ simulation framework Network Definition File (NED) to analyse networks as an extension of the Palladio approach. The utilisation of OMNeT++ provides comprehensive network simulation capabilities, enabling the examination of the impact of network effects on the software system modelled on architectural level. OMPCM integrates the OMNeT++-based network simulation with the Palladio architecture-level software performance prediction to enrich Palladio by more detailed network simulation. OMPCM applies *composition by co-simulation* [20] by having a dedicated bridge to manage the translation of events between the OMPCM and the network simulators. The OMPCM incorporates a series of model transformations, which automatically translate a PCM model into a OMNeT++ NED, utilising the OMPCM modules specifically developed for this purpose.

PCA: The Power Consumption Analyzer (PCA) [36] is another extension that uses the results of Palladio's SimuLizar performance analysis to forecast power consumption of software systems. The Power Consumption metamodel and the performance analysis results of SimuLizar are combined to reason about power consumption on the architecture level. The PCA supports both static and self-adaptive software systems, using measurements from the Palladio Runtime Measurement Model and a stateful Power State Model. The results are accessible in the Palladio Runtime Measurement Model and can trigger self-adaptations in SimuLizar. In PCA, analysis composition is enabled by *result exchange between isolated analysis techniques* [20].

IntBIIS: The performance prediction approach Integrated Business IT Impact Simulation (IntBIIS) [22] extends the PCM and the Palladio performance analysis by a modeling language and analysis technique to predict performance properties of business processes and to investigate the alignment of business processes and software systems with respect to performance. IntBIIS is an example of *composition by extension* [20]. Applying composition by extension in IntBIIS is possible as both, Palladio and the business process extension, adhere to the same modelling paradigm and analysis formalism.

Multi-level hardware simulation: The integration of more detailed hardware simulations in Palladio is another example that demonstrates the need for composing model-based analyses. To enrich Palladio's limited hardware simulation capabilities current work is extending Palladio's software architecture simulation by one or more detailed hardware simulators. The goal is to offer a multi-level hardware simulation to allow the modelling and simulation of a system simultaneously on different levels of abstraction. This allows to switch between these different levels of abstraction, depending on the required trade-off between simulation runtime and result accuracy. Both switching between simulation runs and switching during a simulation run may be supported. Therefore, a composition approach for multiple independent hardware simulators is required that will replace the current hardware simulation capabilities of Palladio in specific use cases that require investigation of detailed hardware properties. The composition of multiple independent hardware simulators uses *composition by result exchange* [20] and is connected to Palladio with the composition operator *composition by co-simulation* [20]. This requires extensions to the PCM as well as to SimuLizar. An example of a model element required on every level of abstraction is the specification of the hardware resource demand, which might be a number of cycles on a high level of abstraction and functional code on a low level. Because such hardware resource demands are a common element of a PCM manually specifying them on multiple levels might not be feasible. Therefore, the second aspect of this approach is the automatic generation and transformation of inputs and outputs between simulations on different levels.

Coupling of architectural analyses and static source code analyses: The coupling of analyses on architectural level and static source code analysis for investigating security properties of the system is another example of a composition relevant in the context of FeCoMASS.

In the input models of architectural security analyses and specification-based static source code analyses, security information is specified for system elements to express their characteristics w.r.t. a security property, e.g., the confidentiality level of data for information flow security. This security information has a type, e.g., the confidentiality level of data, and a range of values, e.g., high and low. However, the actual values realized in the implementation may not conform to the assigned values in the specifications, e.g. by implementing an illegal flow from high data to low data. While the architectural analyses has to assume that the specified security information are realized in the implementation, source code analyses can detect such non-conformances. A coupling approach could therefore comprise an alignment of the assumed values in an architectural analysis input model and the actual values realized in the implementation, obtained by a static source code analysis, for security information of the same type. For this approach, however, the specified security information of the system elements w.r.t. the security property of interest and their values have to be the same in the input models of the architectural analyses and source code analyses. The coupling approach, therefore, comprises three steps: 1. Alignment of the system representations and specifications in the input models of the

architectural analysis and the static source code analysis w.r.t. an analysed security property, e.g., secure information flow. 2. Executing the static source code analysis, which provides indications of violations of the specification of the source code by the implementation. Because of the alignment of step 1), these violations are non-conformances of the implementation to the architectural specification. 3. Extraction of the effective security-related information in the implementation w.r.t. the analyzed property from the source code analysis result and their integration into the architectural analysis input model. Consequently, this scenario applies *composition by result exchange* [20]. With this coupling, the architectural analysis performs its prediction with the information realized in the implementation rather than the assumption that a specified security-related information of an architectural element is correctly implemented. Architectural analysis on which this approach can be applied are for example those of Kramer et al. [28] and Seifermann et al. [35] which both use the PCM as a modeling language. Examples for source code analyses which are applicable in this approach are KeY [2], JOANA [15] or CodeQL [1]

3.2 Camunda – Business Process Analysis

Camunda BPM[4] is an open-source platform for workflow and business process management. In 2013 Camunda BPM was forked from the workflow management system Activiti as an open-source project. Camunda BPM provides a web-based process modelling environment, an execution engine to run the business processes, and a set of tools for monitoring and managing the business process execution. The Camunda BPM platform shows a size of over 500,000 lines of code organised in around 7,000 classes and 23 projects.

Camunda BPM allows organisations to analyse their business processes modelled using the Business Process Modelling Notation 2 (BPMN2) modeling language. The BPMN2 modeling language has been applied as a case study for the decomposition and composition of modeling languages in our previous work [23]. In FeCoMASS, we focus on the analysis techniques of the Camunda BPM platform. Due to the ten years of evolution history of Camunda BPM and the dependencies of the BPMN2 modeling language and the analysis techniques for business process analysis, Camunda BPM seems to be an interesting case for further investigation on the decomposition and composition of model-based analyses in FeCoMASS.

3.3 KAMP4aPS – Change Propagation Analysis for Automated Production Systems

Karlsruhe Architectural Maintainability Prediction for Automated Production Systems (KAMP4aPS) [21] is an approach to model automated production systems and predict the impacts of changes in these systems. KAMP4aPS is under development since 2016. KAMP4aPS is an instantiation of the Karlsruhe

[4] Camunda github: `https://github.com/orgs/camunda/repositories`

Architectural Maintainability Prediction (KAMP) methodology [17]. The KAMP methodology has been developed to provide a blueprint for architecture-based change impact analysis in various domains and has been instantiated among others for software systems [33], business processes [32], production systems [21], and Programmable Logic Controller (PLC) software [8].

KAMP4aPS comprises around 700 classes organised in 54 packages. The modeling language of KAMP4aPS has been applied as a case study for the decomposition and composition of modeling languages in our previous work [23]. In FeCoMASS, we focus on the analysis techniques of KAMP4aPS. KAMP4aPS seems to be an interesting case for further investigation of the decomposition and composition of model-based analysis in FeCoMASS due to (a) the different domains — mechanics, electrics/electronics and software — involved in automated production systems, (b) the dependencies of modeling languages and analysis techniques in KAMP4aPS, and (c) the instantiation of the more general KAMP methodology for change impact analysis.

3.4 SmartGrid Topology – Resilience Analysis of Energy Networks

The SmartGrid Topology [29] analysis approach is used for impact and resilience investigation for smart grid topologies. Its development started in January 2014 and was initially released in October 2015. SmartGrid Topology comprises about 280 classes organised in 34 packages.

The SmartGrid Topology modeling language consists of four views: the topology view, the device types view, the input state view, and the output state view. Input and output state views are implemented in their own language components. The SmartGrid Topology modeling language has been applied as a case study for the decomposition and composition of modeling languages in our previous work [23]. In FeCoMASS, we focus on the analysis techniques of the SmartGrid Topology approach. In contrast to SimuLizar, for example, the SmartGrid Topology approach is more stable and more modular, and covers a different domain (energy networks) and a different quality property (resilience). This makes it an interesting case for further investigation of the decomposition and composition of model-based analyses in FeCoMASS.

3.5 Internet of Production – Aggregated Error Analysis

The excellence cluster Internet of Production (IoP) investigates novel methods of connecting, controlling and monitoring industrial machinery. One of the case studies is a robotic arm with multiple axes of freedom (cf. fig. 2). In order to precisely move such a robotic arm the control logic requires a good understanding of the system. For example, small production errors in the components or friction lead to imprecise movements. To understand how these errors of subcomponents influence the whole system, all components and their interconnection via joints are modelled in SysML. Since the robotic arm consists of several similar components these can be modeled individually and the model of the whole system can then be composed of the smaller components.

Fig. 2. Symbolic Picture of Robotic Arm

Fig. 3. Model of Robotic Arm

Figure 3 depicts how such a a system might be modelled. Each joint receives control information from the previous joint while their respective positions influence each other. Given both models for the individual components and the composite system we aim to investigate how an analysis of the individual components can be composed to obtain an analysis of the entire system. Given individual analyses of these imprecisions we aim at composing these to obtain the error of the entire system.

3.6 Internet of Production – Analysis of Effect Chains

Another case study in the excellence cluster Internet of Production is an injection molding machine. Such machines are widely used in the mass production of plastic parts. The basic working principle of these machines is that plastic can be melted and then injected into a mold, essentially a 3-dimensional negative of the part to be produced, and then cooled down and the solidified plastic can then be ejected. An injection molding machine consists of several parts in general. The process begins with a granulate which is dried in a dryer. Then the granulate is melted and compressed to remove any air in an injector. The injector then injects the molten plastic into the mold with high pressure. The mold is clamped shut using electric motors, hydraulics or magnets. Since the produced part can be ejected only when it sufficiently solidified the mold is often cooled actively. When the part is ready the clamp is released and ejected using different means like pressurised air, mechanical ejectors or robotic arms. A basic model of an injection molding machine might consist of a dryer, an injector and a clamping unit. Each

of those components can be decomposed into various sensors to measure humidity, temperature and pressure, control units, motors and actuators.

With regards to a system like an injection molding machine one might want to analyse performance, verify functionality and understand the system and the effects the various components have on other parts of the system. In order to facilitate these analyses this system or a subsystem can be modelled using components and connectors. Depending on the intended analysis this allows us to model not only logical flow of information but also physical functionalities of the system. The underlying semantic in FOCUS allows us to represent the connections between components as channels with streams which can represent data, material and energy. In order to analyse effects of components on others, and in this context particularly performance, we need to extend FOCUS and investigate using effect chains.

4 Conclusion

In this technical report, we presented scenarios for the decomposition and composition of (modular) model-based analyses for different quality properties and different domains to serve as a basis for illustrative examples and evaluation scenarios in the FeCoMASS project.

Based on these scenarios we will investigate (i) semantic foundations of analysis techniques with a special focus on how compositionality of semantics can be transferred to composition of analysis techniques; (ii) how to transfer these semantic foundations into concepts and detailed guidelines for the (de)composition and extension of model-based analyses and develop composition operators; (iii) how to manage the interactions between analysis features for given composition specifications. We will investigate semantically well-founded analysis interaction points as the glue to combine analysis techniques to higher system understanding.

References

1. CodeQL, `https://codeql.github.com/`
2. Ahrendt, W., Beckert, B., Bubel, R., Hähnle, R., Schmitt, P.H., Ulbrich, M. (eds.): Deductive software verification - the keY book: from theory to practice. No. 10001 in Lecture notes in computer science, Springer, Cham (2016), oCLC: 970816821
3. Bause, F., Buchholz, P., Kemper, P.: Qpn-tool for the specification and analysis of hierarchically combined queueing petri nets. In: Beilner, H., Bause, F. (eds.) Quantitative Evaluation of Computing and Communication Systems. pp. 224–238. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
4. Becker, M., Luckey, M., Becker, S.: Performance analysis of self-adaptive systems for requirements validation at design-time. In: Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures. p. 43–52. Association for Computing Machinery, New York, NY, USA (2013). `https://doi.org/10.1145/2465478.2465489`, `https://doi.org/10.1145/2465478.2465489`
5. Brosch, F., Koziolek, H., Buhnova, B., Reussner, R.: Architecture-based reliability prediction with the palladio component model. IEEE Transactions on Software Engineering **38**(6), 1319–1339 (2012)

6. Broy, M., Rumpe, B.: Modulare hierarchische Modellierung als Grundlage der Software- und Systementwicklung. Informatik Spektrum **30**(1), 3–18 (2007). https://doi.org/10.1007/s00287-006-0124-6
7. Broy, M., Stølen, K.: Specification and Development of Interactive Systems. Focus on Streams, Interfaces and Refinement. Springer (2001)
8. Busch, K., Rätz, J., Koch, S., Heinrich, R., Reussner, R., Cha, S., Vogel-Heuser, B.: A metamodel-based approach to calculate maintainability task lists of plc programs for factory automation. In: 44th Annual Conference of the IEEE Industrial Electronics Society (IECON). IEEE (2018). https://doi.org/10.1109/IECON.2018.8591302
9. Butting, A., Eikermann, R., Kautz, O., Rumpe, B., Wortmann, A.: Systematic Composition of Independent Language Features. Journal of Systems and Software **152**, 50–69 (2019). https://doi.org/10.1016/j.jss.2019.02.026
10. Combemale, B., Kienzle, J., Mussbacher, G., Barais, O., Bousse, E., Cazzola, W., Collet, P., Degueule, T., Heinrich, R., Jézéquel, J.M., et al.: Concern-oriented language development (cold): Fostering reuse in language engineering. Computer Languages, Systems & Structures **54**, 139–155 (2018)
11. Czarnecki, K., Eisenecker, U.W.: Generative Programming. Addison-Wesley (2000)
12. Durán, F., Heinrich, R., Pérez-Palacín, D., L. Talcott, C., Zschaler, S.: Composing Model-Based Analysis Tools (Dagstuhl Seminar 19481). Dagstuhl Reports **9**(11), 97–116 (2020). https://doi.org/10.4230/DagRep.9.11.97
13. Fowler, M.: Domain-specific languages. Pearson Education (2010)
14. Friedenthal, S., Moore, A., Steiner, R.: A practical guide to SysML: the systems modeling language. Morgan Kaufmann (2014)
15. Graf, J., Hecker, M., Mohr, M., Snelting, G.: Sicherheitsanalyse mit joana. In: Sicherheit 2016: Sicherheit, Schutz und ZuverlÄssigkeit, BeitrÄge der 8. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft fÜr Informatik e.V. (GI), 5.-7. April 2016, Bonn. pp. 11–22 (2016). https://doi.org/20.500.12116/869
16. Harel, D., Rumpe, B.: Meaningful Modeling: What's the Semantics of "Semantics"? IEEE Computer **37**(10), 64–72 (2004). https://doi.org/10.1109/MC.2004.172
17. Heinrich, R., Busch, K., Koch, S.: A methodology for domain-spanning change impact analysis. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 326–330. IEEE Computer Society (2018). https://doi.org/10.1109/SEAA.2018.00060
18. Heinrich, R.: Architecture-based Evolution of Dependable Software-intensive Systems. KIT Scientific Publishing (2023)
19. Heinrich, R., Durán, F., Talcott, C.L., (eds.), S.Z.: Composing Model-Based Analysis Tools. Springer (2021)
20. Heinrich, R., Henss, J., Koch, S., Reussner, R.: Challenges in the Evolution of Palladio—Refactoring Design Smells in a Historically-Grown Approach to Software Architecture Analysis, pp. 235–257. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-81915-6_11, https://doi.org/10.1007/978-3-030-81915-6_11
21. Heinrich, R., Koch, S., Cha, S., Busch, K., Reussner, R., Vogel-Heuser, B.: Architecture-based change impact analysis in cross-disciplinary automated production systems. Journal of Systems and Software **146**, 167 – 185 (2018). https://doi.org/https://doi.org/10.1016/j.jss.2018.08.058, https://doi.org/10.1016/j.jss.2018.08.058
22. Heinrich, R., Merkle, P., Henss, J., Paech, B.: Integrating business process simulation and information system simulation for performance prediction. Software & Systems Modeling **16**, 257–277 (2017)

23. Heinrich, R., Strittmatter, M., Reussner, R.H.: A layered reference architecture for metamodels to tailor quality modeling and analysis. IEEE Transactions on Software Engineering (2019). https://doi.org/10.1109/TSE.2019.2903797
24. Heinrich, R., Werle, D., Klare, H., Reussner, R., Kramer, M., Becker, S., Happe, J., Koziolek, H., Krogmann, K.: The palladio-bench for modeling and simulating software architectures. p. 37–40. ICSE '18, Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3183440.3183474, https://doi.org/10.1145/3183440.3183474
25. Henss, J., Merkle, P., Reussner, R.H.: The ompcm simulator for model-based software performance prediction. In: Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques. pp. 354–357 (2013)
26. Hölldobler, K., Kautz, O., Rumpe, B.: MontiCore Language Workbench and Library Handbook: Edition 2021. Aachener Informatik-Berichte, Software Engineering, Band 48, Shaker Verlag (May 2021), http://www.monticore.de/handbook.pdf
27. Hölldobler, K., Rumpe, B., Wortmann, A.: Software language engineering in the large: towards composing and deriving languages. Computer Languages, Systems & Structures **54**, 386–405 (2018). https://doi.org/10.1016/j.cl.2018.08.002
28. Kramer, M.E., Hecker, M., Greiner, S., Bao, K., Yurchenko, K.: Model-Driven Specification and Analysis of Confidentiality in Component-Based Systems. Tech. rep., Karlsruhe Institute of Technology, Department of Informatics, Karlsruhe (2017). https://doi.org/10.5445/IR/1000076957, http://dx.doi.org/10.5445/IR/1000076957
29. Ottenburger, S.S., Münzberg, T., Strittmatter, M.: Smart grid topologies paving the way for an urban resilient continuity management. International Journal of Information Systems for Crisis Response and Management (IJISCRAM) **9**(4), 1–22 (2017). https://doi.org/10.4018/IJISCRAM.2017100101
30. Reussner, R.H., Becker, S., Happe, J., Heinrich, R., Koziolek, A., Koziolek, H., Kramer, M., Krogmann, K.: Modeling and Simulating Software Architectures – The Palladio Approach. MIT Press (2016)
31. Ringert, J.O., Rumpe, B.: A Little Synopsis on Streams, Stream Processing Functions, and State-Based Stream Processing. International Journal of Software and Informatics pp. 29–53 (2011)
32. Rostami, K., Heinrich, R., Busch, A., Reussner, R.: Architecture-based Change Impact Analysis in Information Systems and Business Processes. In: International Conference on Software Architecture. pp. 179–188. IEEE (2017), https://doi.org/10.1109/ICSA.2017.17
33. Rostami, K., Stammel, J., Heinrich, R., Reussner, R.: Architecture-based assessment and planning of change requests. In: Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures. p. 21–30. QoSA '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2737182.2737198, https://doi.org/10.1145/2737182.2737198
34. Rumpe, B.: Agile Modeling with UML: Code Generation, Testing, Refactoring. Springer (2017)
35. Seifermann, S., Heinrich, R., Werle, D., Reussner, R.: Detecting violations of access control and information flow policies in data flow diagrams. Journal of Systems and Software **184**, 111138 (2022). https://doi.org/https://doi.org/10.1016/j.jss.2021.111138, https://www.sciencedirect.com/science/article/pii/S0164121221002351
36. Stier, C.: Adaptation-Aware Architecture Modeling and Analysis of Energy Efficiency for Software Systems (2018). https://doi.org/10.5445/IR/1000083402

37. Strittmatter, M.: A Reference Structure for Modular Metamodels of Quality-Describing Domain-Specific Modeling Languages. Ph.D. thesis, Karlsruhe Institute of Technology (KIT) (2020). https://doi.org/10.5445/KSP/1000098906
38. Strittmatter, M., Hinkel, G., Langhammer, M., Jung, R., Heinrich, R.: Challenges in the evolution of metamodels: Smells and anti-patterns of a historically-grown metamodel. In: 10th International Workshop on Models and Evolution (ME). CEUR Vol-1706 (2016), http://ceur-ws.org/Vol-1706/
39. Talcott, C., Ananieva, S., Bae, K., Combemale, B., Heinrich, R., Hills, M., Khakpour, N., Reussner, R., Rumpe, B., Scandurra, P., Vangheluwe, H., Durán, F., Zschaler, S.: Foundations, pp. 9–37. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-81915-6_2, https://doi.org/10.1007/978-3-030-81915-6_2
40. Völter, M., et al.: Model-driven software development: technology, engineering, management. Wiley (2013)
41. Walter, M., Heinrich, R., Reussner, R.: Architecture-based attack path analysis for identifying potential security incidents. In: 17th European Conference on Software Architecture (ECSA) (2023)
42. Zeigler, B.P., Muzy, A., Kofman, E.: Theory of Modeling and Simulation: Discrete Event and Iterative System Computational Foundations. Academic Press, Inc., USA, 3rd edn. (2018)