A Methodology for Domain-spanning Change Impact Analysis

Robert Heinrich, Kiana Busch, Sandro Koch

Karlsruhe Institute of Technology
Karlsruhe, Germany
Email: {robert.heinrich, kiana.busch, sandro.koch}@kit.edu

***Abstract*—When modifying a cyber-physical system, the consequences of changes need to be understood beforehand to adequately assess risks and costs. Model-based change impact analysis is key for estimating the impact of a change before actually modifying the system. Existing change impact analysis approaches apply very similar algorithms for change propagation to instances of domain-specific metamodels. However, they lack fundamental concepts for domain-spanning change impact analysis. In this paper, we propose a generic methodology for domain-spanning change impact analysis to address limitations of existing approaches. Evaluation results show the relevancy and comprehensives of the methodology for several domains.**

***Index Terms*—Change Impact Analysis, Reuse, Consistency**

I. Introduction

As cyber-physical systems are long-living systems, maintainability is one of their most important quality aspects. Maintainability is commonly understood as the capability of a system to be changed [1]. Thus, maintainability refers to the efforts required for implementing changes in the system. While changing a system, the impact of a change to one artifact on the other artifacts need to be understood. Different ways of implementing a change request may lead to different efforts in adapting the system. Estimating the effort before implementing a certain change request is essential to make decisions in which way to implement changes [2].

For representing a cyber-physical system in form of a model, a modeling language (often defined by a metamodel) is required. A metamodel is a model, which defines the structure and characteristics of other models. Model-based change impact analysis is key for maintainability assessment in various domains. These approaches allow for automatic change propagation in a system. Existing approaches to change impact analysis apply very similar algorithms for change propagation to instances of domain-specific metamodels. However, they are often restricted to a specific domain [2], [3], [4].

In this paper, we propose a generic methodology for domain-spanning change impact analysis to address limitations of existing approaches. We generalize the approach Karlsruhe Architectural Maintainability Prediction (KAMP) for architecture-based change impact analysis in different domains (cf. [2], [3]) by extracting commonalities in modeling and analysis into the methodology. The methodology is part of a broader research vision proposed in [5]. Contributions of this paper are as follows: i) The methodology represents a generic frame, which can be instantiated to specify domain-specific change impact analysis approaches. Thus, it facilitates the creation of domain-specific change impact analysis by providing basic concepts in terms of metamodels and algorithms. The metamodels and algorithms can be reused and tailored for change impact analysis in specific domains. ii) The methodology ensures compatibility of domain-specific analyses, as they rely on the same fundamental concepts. Thus, the methodology supports domain-spanning change impact analysis. iii) We instantiate the methodology to different domains. Each instantiation automatically predicts the impact of a change request in a given system model. Evaluation results show the relevancy and comprehensiveness of the methodology for the instantiations.

The following section discusses the state of the art. The methodology for domain-spanning change impact analysis is given in Sec. III. Sec. IV describes the evaluation of the methodology. The paper concludes in Sec. V.

II. State of the Art

The state of the art related to the topic of the paper comprises change impact analysis approaches in the domain of Information Systems (IS), Business Processes (BP), and automated Production Systems (aPS), which are listed in [2], [3], and [4], respectively. However, most of the related approaches are limited to a certain domain and mostly neglect the impact of changes in one domain on another, which is crucial for adequate change impact analysis. When applying existing approaches to different domains it is hard to compare and aggregate the results due to the heterogeneity of the approaches for modeling and change propagation analysis in the different domains. A more generic approach for domain-spanning change impact analysis is required to address limitations of existing approaches.

III. A Generic Methodology for Domain-spanning Change Impact Analysis

In this section, we propose the methodology for domain-spanning change impact analysis to address the limitations of existing approaches. The methodology represents a generic frame defining the domain-independent concepts that need to be instantiated to define domain-specific change impact analysis. The input of a domain-specific change impact analysis is a system model and an initial change, hereafter referred to as *seed modification* [2]. Based on the input and a set of

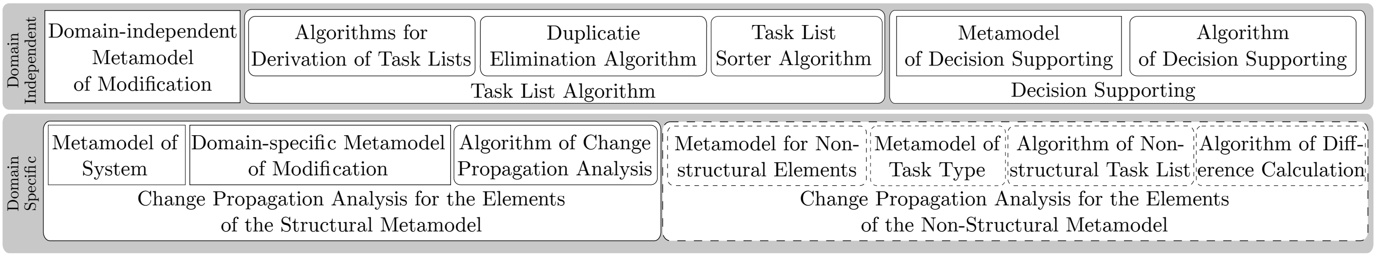
Domain Independent
Domain-independent Metamodel of Modification
Algorithms for Derivation of Task Lists
Duplicatie Elimination Algorithm
Task List Sorter Algorithm
Task List Algorithm
Metamodel of Decision Supporting
Algorithm of Decision Supporting
Decision Supporting
Domain Specific
Metamodel of System
Domain-specific Metamodel of Modification
Algorithm of Change Propagation Analysis
Change Propagation Analysis for the Elements of the Structural Metamodel
Metamodel for Non-structural Elements
Metamodel of Task Type
Algorithm of Non-structural Task List
Algorithm of Difference Calculation
Change Propagation Analysis for the Elements of the Non-Structural Metamodel

Fig. 1: Overview of the methodology for the domain-spanning change propagation analysis

predefined *change propagation rules*, the methodology instance calculates a list of maintenance tasks, hereafter called *task list* [2], [6]. Each task references an element of the domain-specific metamodel and has a task type [6]. A example of a task is "delete interface I". A task type describes how to change the corresponding element [6]. Here, the task type is "DELETE" and the affected element is "interface I".

The methodology generalizes the KAMP approach for any domain, which fulfills the following criteria: (i) The system can be described as a set of structural elements connected to each other. For example, in a component-oriented fashion with interfaces and component composition. By adhering to this basic paradigm the methodology can be applied regardless of the type of component (e.g., software, electrical, or mechanical component) and the type of interface (e.g., communication or physical interface). (ii) The behavior of the system can be described in terms of interconnected activities and their compositions. Thus, the methodology can be applied despite of whether the activity is performed by a software, mechanical system, or others. (iii) The change propagation between the elements of the system and/or activities of its behavior can be represented as change propagation rule(s).

The structure and behavior of the system, as well as changes to them are defined by a metamodel. Moreover, algorithms handle the change propagation. A change propagation algorithm consists of a set of change propagation rules. An overview of the generic methodology is depicted in Fig. 1. Rectangles with tapered corners represent metamodels, while rectangles with rounded corners represent algorithms that work on instances of the metamodel elements. The core elements of the methodology are independent of a specific domain and can be used in any instance of the methodology. Domain-specific elements consist of metamodels and algorithms to describe the specific characteristics of a given domain.

A. Domain-Independent Elements

Domain-independent elements of the methodology comprise algorithms for assembling the task list and metamodels for specifying changes. In the implementation of the methodology, the system is represented as generics at this level and further specified in the domain-specific metamodels. In the following, we discuss the domain-independent elements in more detail.

1) Domain-Independent Metamodel of Modification: Model-based change propagation analysis in different domains has the following metaclasses in common: i) A set of all seed modifications. ii) A set of all model elements that may be potentially affected by a change. In a change propagation between two elements, one element is the cause of the change, hereafter referred to as *causing element*, while the other element is affected by the change caused by the first element, hereafter referred to as *affected element*. iii) Change propagation from affected elements to further elements, hereafter referred to as *change propagation step*. iv) The task list involves seed modifications and a set of change propagation steps. This is represented by the *modification repository* metaclass in the domain-independent metamodel of modification. An instantiation of the methodology for a specific domain has to extend the aforementioned metaclasses.

2) Task List Algorithms: Besides the metamodels of modification, the change propagation algorithms play a central role in the change propagation analysis. These algorithms use instances of the metamodels to derive the task lists. In the following the main algorithms are described:

Algorithm for Derivation of Task Lists: Central to the methodology is an algorithm to derive task lists based on domain-specific models. This algorithm calculates the differences between two models: *base version* and *target version*. While the *base version* represents the model before the change, the *target version* represents the model after the change. Given two models, the differences between both can be identified by a model diff. Further, the algorithm extends the task list containing the changes to structural elements to include tasks for maintaining the non-structural elements. Then, it calculates the task list, removes the duplicates, and sort the task list in order to have deterministic task lists by the same input.

Algorithm for Duplicate Elimination: The derived task lists may contain duplicated tasks referring to the same model element. Such tasks may be generated in different change propagation iterations by different change propagation rules, or due to several seed modifications [3]. Further, a task can have subtasks. For example, the task "modify interface I" can have the subtask "modify the corresponding signatures" [6]. Additionally, a task can have follow-up tasks. In the aforementioned example, the task "modify interface I" can have the follow-up task "update test cases" [6]. Thus, we have to consider the subtasks and follow-up tasks while merging the tasks and removing duplicates. This algorithm recursively merges all tasks that refer to the same element and have the same type.

Algorithm for Task List Sorter: If the same change requests affect the system, the results must be deterministic. Further, to improve the comparability between the task lists, the results should be in the same order [3]. This algorithm recursively sorts the task list while considering top-level tasks (e.g., modify interface I), as well as subtasks (e.g., modify the corresponding signatures), and follow-up tasks (e.g., update test cases).

3) Decision Support: It may be necessary to include the knowledge of a domain expert not contained in the models to reason about the task list. Domain experts may be software architects in the domain of IS, process designers in the domain of BP, or system engineers in the domain of aPS. For example, if the system is composed of elements that cannot be changed (e.g., third party components), it is not reasonable to have them in the task list. The involvement of domain experts is also required in situations where a false positive in one of the previous iterations propagates to more false positives in the subsequent iterations. Therefore, we enable the domain experts to mark model elements that cannot to be changed. Based on their decision, the task list is recalculated and the elements that are excluded due to expert knowledge are eliminated [3].

Metamodel of Decision Support: This metamodel includes three basic types of human decision on a task: `Confirm`, `Exclude`, and `No Decision` as enums. `No Decision` is the default decision, after the task list is generated. The domain expert can mark tasks that cannot be changed as `Exclude`. The true positive tasks in the task list can be marked as `Confirm`.

Algorithm of Decision Support: An `Excluded` task can result in follow-up tasks that also have to be `Excluded`. The first iteration excludes elements that are marked by the human. These elements can be the causing elements for further elements. In each iteration, elements that do not have a causing element are removed. These elements are the newly excluded elements. The algorithm terminates, if there are not any elements in the set of all excluded elements.

B. Domain-Specific Elements

The instantiation of the methodology for a specific domain requires the extension of the domain-specific part of the methodology. The domain-specific elements of the methodology can be divided into two parts. The first part refers to metamodels and algorithms mandatory for analyzing the architecture-based change propagation and for deriving a task list. The second part refers to elements that are optional in change propagation analysis. They are illustrated in Fig. 1 by dashed rectangles.

1) Change Propagation Analysis for the Elements of the Structural Metamodel: The mandatory part of the methodology defines the extension points needed for instantiating the methodology to a specific domain. It comprises a metamodel for describing the structure of the system and a metamodel for specifying the change propagation, hereafter called *domain-specific metamodel of modification*. In addition, the domain-specific change propagation algorithm is required. Based on an instance of the domain-specific metamodel of modification, the metamodel of the system, and the corresponding change propagation algorithm, the instance of the methodology identifies the model elements affected by a given seed modification.

Metamodel of System: For instantiating the methodology to a specific domain the metamodels for describing the systems in this domain must be created. In the following, we discuss model elements relevant for change propagation that should be considered while constructing domain-specific metamodels.

Structure: The structure of the system under study plays a key role in the change propagation analysis [7]. The domain expert has to metamodel the structure of the systems in the domain. For example, the system can be structured using components and their compositions as common in architectural description languages like SysML [8] and Palladio [9].

Data Flow: Another important aspect of the change propagation analysis is the propagation of changes by the data flow, as changes may propagate due to data dependencies [10], [7]. The representation of a data is domain-specific. For example, in the domain of IS, the data types can represent the data flow [9], whereas signal input or output can represent the data flow in an aPS [11]. In the BP a data may be represented by a physical data object [3]. Consequently, a change can propagate between two domains using the data flow, as different representations of the data can be converted in each other.

Behavior: The behavior may also be relevant for the change propagation, as a change to the structure of a system or the data flows can affect the behavior of the system [7]. For example, a change to the software can propagate to its users' interface. Thus, a software change may affect the users' experience [12].

Domain-specific Metamodel of Modification: This metamodel is an extension of the domain-independent metamodel of modification for a specific domain. This extension involves specifying the seed modifications, potentially affected elements by changes, and the common causes for change propagation steps (e.g., data dependencies) [2]. Further, the modification repository has to be extended by the domain-specific seed modifications and change propagation steps.

Algorithm of Change Propagation Analysis: Each rule of the change propagation algorithm is specified for the elements of the metamodel of the system and the domain-specific metamodel of modification. The domain expert can specify the change propagation rules in a general-purpose programming language like Java. Alternatively, the rules can be specified in our Change Propagation Rule Language (CPRL) [13]. CPRL is a declarative language for expressing the change propagation along the references of the metaclasses (i.e., forward, backward, and their combinations). While the focus of this paper is on the metamodels, details on the analysis algorithms are described in our previous work [2], [3], [4], [14].

2) Domain-Specific Change Propagation Analysis for Non-structural Metamodel Elements: Implementing a change request may additionally involve organizational or technical tasks [2]. An example for such tasks is adapting or executing test cases in IS [2]. This part of the methodology extends the task list by non-structural elements. As non-structural elements might not be needed in all domains, the metamodels and algorithms in this part of the methodology are considered as optional. In the following, more details on the metamodels and algorithms for non-structural elements are given.

Metamodel of Non-structural Elements: This metamodel includes various element types that are not part of the structure of the system. These elements can be, however, relevant for the change propagation analysis in a specific domain [2], [4]. As the metamodel of the system acts as the main artifact for the change propagation analysis, the non-structural elements refer to the metamodel of the system. For example in the IS domain, the "test case" can refer to the "interface" metaclass [6].

Metamodel of Task Type: This metamodel represents the domain-specific task types such as writing test cases [6]. Each task type also refers to a non-structural elements. As not all non-structural elements may have a domain-specific task type, this metamodel can be considered as optional.

Algorithm of Non-structural Task List: If there are non-structural elements in the domain under study, the change propagation analysis estimates how they are affected by a given change request. For example, if an interface, which has some test cases, is removed, the test cases have to be removed, too. In contrast, if the interface is modified, the corresponding test cases may need to be updated and executed [2]. As this highly depends on the domain under study, this algorithm has to be implemented based on the domain-specific metamodels and the change propagation rules.

Algorithm of Difference Calculation: During a modification, it may be necessary to change the way in which the structural elements of the system are connected to each other. Further, new elements may be added to, or elements may be removed from the model of the system. Thus, a further aspect is the change propagation analysis due to changes to the structure of the system model. After we changed the structure of the system in a specific domain, the algorithm identifies the differences between two versions of the system model (i.e., before and after the change) [2]. This comparison allows for deriving added or removed elements in addition to changed elements.

IV. Evaluation

In this section, we instantiate the methodology to several domains to evaluate its relevancy and comprehensiveness.

A. Evaluation Design

The **Goal** of the evaluation is to show the methodology is sufficiently relevant and comprehensive when instantiated for different domains. For evaluating the methodology we analyze several domain-specific instantiations[1]. It is important to note that the goal of the evaluation is not to show the quality of the automatically generated task lists of the individual instances. One reason for this is that the methodology can be instantiated in a certain domain in different ways, which influence the quality of the generated task lists of individual instances. Thus, there is no unique instance of the methodology in a certain domain. Further, the quality of the individual instances has already been evaluated in previous studies [2], [3], [4], [14].

For evaluating the relevancy, we define **Question 1**: In how many domain-specific instantiations of the methodology are the elements of the methodology used? The more domain-specific instantiations use the elements of the methodology, the more relevant we consider the elements. For evaluating the coverage of the elements common for several instantiations, we define **Metric 1** as the ratio, R, of the number of instantiations of the methodology in which an element occurs, U, to the number of all instantiations of the methodology, N. Thus, we can calculate Metric 1 as $R = \frac{U}{N}$.

Furthermore, we evaluate the comprehensiveness of the methodology by **Question 2**: Are there any common metamodels or algorithms in the several instantiations, that are not specified by the methodology? In the evaluation we consider a metamodel or an algorithm as "common", if it is present in more than one instantiation of the methodology. When instantiating the methodology in a new domain, this research question investigates, whether there are new metamodels or algorithms for change propagation analysis in this domain that have not yet been considered in the methodology. The less common metamodels or algorithms missing in the methodology we identify, the more comprehensive we consider the methodology. To answer this question, we define **Metric 2** as the number of common metamodels or algorithms in the domains that are not specified by the methodology.

In the following, we describe the selected domains, which fulfill the criteria for instantiating the methodology regarding the system's structure and its behavior, as well as the change propagation between its elements (cf. Sec. III):

We chose the domain of BP, as a BP can be defined as a set of actor steps and system steps [15]. Thus, a change to the IS can affect the system step and thus the entire BP [3].

An aPS involves electronic, mechanical, and software parts. All parts represent the heterogeneous structure of the aPS. An automatic change propagation approach is appropriate for analyzing the heterogeneous structure. The instantiation of the methodology for aPS can be found in [4].

The initial change can be specified either at system element or at requirement level. In the first case, the domain experts must select the initially affected system elements based on the change request. In the second case, a change in a requirement triggers the change propagation. Although requirements are not a stand-alone domain, it is often helpful to consider requirement changes as they may trigger changes in the systems implementing them. Details on the methodology instantiation for requirements in the aPS domain are given in [14].

B. Evaluation Results

Tab. I summarizes the results of the evaluation for **Question 1** and **Question 2**. To answer **Question 1**, we investigated for all metamodels and algorithms of the methodology whether they occur in the instantiations of the methodology after implementing them. The rows in the third column of Tab. I present the elements of the methodology. Each row in the columns with headers IS, BP, aPS Hardware, aPS Software, and Req shows, whether the corresponding instantiations for these domains implement the elements of the methodology. The number of instantiations of the methodology (i.e., N) is 5. The last column of Tab. I presents Metric 1.

The domain-independent metamodels and algorithms were required in all domains. Thus, the number of instantiations of

[1]https://github.com/KAMP-Research/

TABLE I: Commonalities of the methodology instantiation. Legend: ✓:used, x : not used

		Domain: / Metamodels & Algorithms:	IS	BP	aPS Hardware	aPS Software	Req	Metric 1
Do -main -inde -pendent		Domain-independent Metamodel of Modification	✓	✓	✓	✓	✓	1.0
	Task List Algorithm	Algorithm for Duplicate Elimination	✓	✓	✓	✓	✓	1.0
		Algorithm for List Sorter	✓	✓	✓	✓	✓	1.0
		Algorithm for Derivation of Task List	✓	✓	✓	✓	✓	1.0
	Decision Supporting	Metamodel of Decision Supporting	✓	✓	✓	✓	✓	1.0
		Algorithm for Decision Supporting	✓	✓	✓	✓	✓	1.0
Do -main -spe -cific	Structural Change Propagation	Metamodel of the System	✓	✓	✓	✓	✓	1.0
		Domain-specific Metamodel of Modification	✓	✓	✓	✓	✓	1.0
		Algorithm for Change Propagation	✓	✓	✓	✓	✓	1.0
	Non-structural Change Propagation Analysis	Metamodel of Non-structural Elements	✓	✓	✓	✓	x	0.8
		Metamodel of Task Type	✓	x	✓	✓	x	0.6
		Algorithm of Non-structural Task List	✓	✓	✓	✓	x	0.8
		Algorithm for Difference Calculation	✓	✓	✓	✓	✓	1.0
Metric 2	Was a common element missing during the instantiation of the methodology?		0	0	0	0	0	

the methodology for these elements (i.e., U) is 5. Consequently, Metric 1 is 1.0 for domain-independent elements. In all domains, we had to create a metamodel describing the system, a domain-specific metamodel of modification, and an algorithm for change propagation. Thus, U is also 5 for these elements. Consequently, Metric 1 is also 1.0 for all mandatory elements of the domain-specific part of the methodology.

The instantiations of the methodology for IS, aPS hardware, and aPS software include all optional elements of the domain-specific part of the methodology. However, we could identify neither metamodels regarding non-structural elements and task types, nor the algorithm of non-structural task list in the instantiation for requirements. For example, there are no test cases for requirements, but for the system elements in a domain to test whether they meet the requirements. The instantiation of the methodology for BP includes the metamodels regarding non-structural elements and the algorithm of non-structural task list. However, we could not identify a task type metamodel for the BP domain. Consequently, Metric 1 is 0.8 for the metamodel of non-structural elements and the corresponding algorithm for deriving the non-structural task list. Metric 1 is 0.6 for the metamodel of task types, as it is not used for BP and requirements. This conforms to our intention to consider these elements of the methodology as optional. Metric 1 is 1.0 for the algorithm of difference calculation, as it could be instantiated in all domains.

If non-structural elements in a domain affect the maintainability analysis of the systems in this domain, the optional elements of the methodology can be used to consider these elements. The optional elements allow a more precise change propagation analysis. As they do not necessarily have to occur in a domain, their non-existence does not affect relevance of the methodology. In conclusion, we consider our methodology as relevant because the evaluation results show all the mandatory metamodels and algorithms of the methodology can be found in domain-specific instantiations of the methodology.

Question 2 addresses metamodels and algorithms missing in the methodology (cf. last row of Tab I). After instantiating the methodology for the respective domain, we could not identify any common metamodel or algorithm that was not specified by the methodology (i.e., Metric 2 is 0). The evaluation results show the methodology is sufficiently comprehensive.

V. Conclusion

This paper proposed domain-independent concepts in form of a generic methodology that need to be instantiated for domain-specific change impact analysis. The methodology addresses limitations of existing approaches by ensuring compatibility of domain-specific analyses, as they rely on the same fundamental concepts. Thus, it supports domain-spanning change impact analysis for the domains of IS, BP, aPS and requirements. Evaluation results show the relevancy and comprehensives of the methodology for domain-specific instantiations. In the future, we will instantiate the methodology to further domains.

Acknowledgment

This work was partially supported by the DFG under SPP1593 (RE1674/12-1) and the MWK in the funding line RiSC. We thank Jakob Bach for inspiring discussions and support during conception and tool development.

References

[1] ISO/IEC, "25010 - Systems and software: Engineering, Quality Requirements and Evaluation, Quality Models," Tech. Rep., 2010.

[2] K. Rostami *et al.*, "Architecture-based assessment and planning of change requests," in *QoSA'15*. ACM, 2015, pp. 21–30.

[3] ——, "Architecture-based Change Impact Analysis in Information Systems and Business Processes," in *ICSA'17*. IEEE.

[4] B. Vogel-Heuser *et al.*, "Maintenance effort estimation with kamp4aps for cross-disciplinary automated production systems - a collaborative approach," in *IFAC*, 2017.

[5] R. Heinrich, "Tailored quality modeling and analysis of software-intensive systems," in *SEKE*, 2018.

[6] J. Stammel, "Architekturbasierte Bewertung und Planung von Änderungsanfragen," Ph.D. dissertation, KIT, 2015.

[7] R. Kazman *et al.*, "Saam: A method for analyzing the properties of software architectures," in *16th ICSE*, ser. ICSE '94. IEEE, pp. 81–90.

[8] OMG, *OMG SysML, Version 1.3*, Object Management Group Std., 2012. [Online]. Available: http://www.omg.org/spec/SysML/1.3/

[9] Reussner, Ralf et al., Ed., *Modeling and Simulating Software Architectures - The Palladio Approach*. MIT Press, 2016.

[10] M. Lee *et al.*, "Algorithmic analysis of the impacts of changes to object-oriented software," in *TOOLS*, 2000, pp. 61–70.

[11] S. Koch, "Automatische Vorhersage von Änderungsausbreitungen am Beispiel von Automatisierungssystemen," Master's thesis, KIT.

[12] N. Chapin *et al.*, "Types of software evolution and software maintenance," *JSM*, vol. 13, no. 1, pp. 3–30, 2001.

[13] K. Busch *et al.*, "A cross-disciplinary language for change propagation rules," in *CASE*. IEEE, 2018.

[14] T. Maier *et al.*, "An approach to requirement analysis in automated production systems," in *20. WSRE – DFF*, 2018.

[15] R. Heinrich *et al.*, "Integrating business process simulation and information system simulation for performance prediction," *SoSyM*, 2015.