Proceedings of the 2019 Winter Simulation Conference
N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C. Szabo, P. Haas, and Y.-J. Son, eds.

TRENDS IN AUTOMATIC COMPOSITION OF STRUCTURES FOR SIMULATION MODELS IN PRODUCTION AND LOGISTICS

Sigrid Wenzel
Jana Stolipin

Institute of Production Technology and Logistics
Dept. of Production Org. and Factory Planning
University of Kassel
Kurt-Wolters-Str. 3
Kassel, 34125, GERMANY

Jakob Rehof
Jan Winkels

Department of Computer Science
Chair for Software Engineering
TU Dortmund University
Otto-Hahn-Str. 12
Dortmund, 44227, GERMANY

ABSTRACT

This paper presents challenges in the field of automatic generation of simulation models in production and logistics. For this purpose, we first present existing approaches in this field, we then analyze them, and we present trends in the field of automatic composition of structures for simulation models. Subsequently, possibilities from the field of software engineering are discussed, illustrating a possible approach to the composition of structures for simulation models. In particular, the focus is placed on synthesis of logical structures by means of combinatory logic. The strategy of how this logic can support the automated generation of structural variants in simulation models is shown on the basis of one of the areas of intralogistics. In our example, different possibilities of structural variants are clarified and their potential for automated composition demonstrated.

1 INTRODUCTION

In the field of production and logistics, simulation is a recognized method of the Digital Factory for evaluation, planning and monitoring of systems and processes (Wenzel et al. 2010; Bracht et al. 2018). In this context simulation is used to ensure planning reliability, to discover potentials for rationalization and to support the decision making process. In production and logistics, tools based on discrete event simulation (DES) paradigms are normally used. Simulation tools allow the software-technical reproduction of systems in DES models and the investigation of system behavior over time. Simulation models are used both in the planning phase and in the operating phase of a production system. In the planning phase, planning concepts are worked out and checked with the help of these models without interfering with the real systems. Process alternatives can be checked, system structures varied and control strategies tried out. In the operating phase, simulation models are used to control the production systems, e.g. as early warning systems (Hotz et al. 2006), to understand complex systems or to support decision-making.

Simulation allows to inspect the effects of changes in production and logistics systems very precisely. The long model building times and high effort, e.g. for collection and analysis of data and information regarding the considered system, complicate the execution of simulation studies in particular with the modeling of complex production systems (Bogon et al. 2012). The quality of the results of a simulation study depends in particular on the quality and scope of the information used in the simulation study (Abel et al. 2013). Within the framework of simulation-based planning of complex production systems, a large number of experiments are carried out (Rabe et al. 2009). Not only the parameters, but also the structures and control strategies of the models have to be varied or adapted. A major challenge in the planning of modern flexible as well as adaptable production and logistics systems is the variation of system structures with simultaneous consideration of the adaptation of corresponding control strategies in the simulation model (Wenzel et al. 2018). Fowler and Rose (2004) discuss challenges for complex production systems, pointing out the need to reduce the effort required to conduct simulation studies and addressing closer cooperation between the real world and the model world. Automatic model generation is proposed as a possible solution, but it is pointed out that existing approaches such as Ozdemirel et al. (1993), Jankauskas and McLafferty (1996) or Law and McComas (1989) will only work if the data are available in the correct form for the simulation model. Approaches to automatic model generation enable efficient implementation of simple standardized models, e.g. using CAD data and work plan data. For example, the automotive industry implements different strategies for semi-automatic or automatic model generation using specialized automotive unit libraries (Jensen 2007; Bergmann and Strassburger 2010; Rooks 2009). These solutions are only useable for manageable standardized systems and do not take into account automatic adjustments of the control strategies of the modeled system. In addition, there is no guarantee that optimal structural variants will be proposed. Most approaches are also time consuming. When planning logistics systems, one is often faced with the challenges of choosing the best and most economical system variant from many possibilities. Therefore, the structure of a system cannot always be considered statically, because it changes over the duration of system existence, which is subsequently understood as structural variance (Tabeling 2006). Tabeling (2006) distinguishes the following kinds of structural variance:

- Modification of the connection structure,
- Emergence and disappearance of components,
- Modifications of component types - internal conversion,
- Structural variance as a targeted conversion of the system,
- Description of dynamic structures by behavioral models,
- Structural variance in programmed systems.

As more and more versatile systems are being used in production and logistics, their digital models must be able to take into account and implement the structural variance of these systems as early as the planning stage. Because many variants are used and experimented with in the planning of systems, the automatic generation of variants of simulation models would reduce the planning effort (Wenzel et al. 2018). Other important challenges in the planning process of the adaptable production and logistics systems are the transparent selection and explanation of the planning alternatives with a view to meaningful target values and their system structures. It must be taken into account that with the increasing number of framework conditions, the number of system variants to be analyzed with different system structures will increase at the same time.

DES offers comprehensive possibilities to plan dynamic processes of flexible and adaptable systems and to test complex scenarios. Even in the age of Industry 4.0, DES plays a key role, which corresponds to the trend towards digital test environments (Schluse, Rossmann 2006; Wenzel et al. 2018). The methods of the Digital Factory can be used to secure the factory system structure (Bracht et al. 2018). In the context of Industry 4.0, process dynamics and product variants will require fast decisions, especially if the scope in decision-making becomes larger in the future and at the same moment the time window for decisions is continuously shrinking (Xu et al. 2016). Xu et al. (2016) discuss optimization by simulation as a suitable decision and analysis tool. In particular, the need for research in the support and verification of simulation-based decisions on comprehensive and flexible systems is pointed out, the models of which can be simulated in real time, e.g. with the help of several simulation computers, and used for decision making.

In future, simulation models will also have to meet requirements such as adaptability and applicability over the entire plant life cycle. These requirements entail a certain conceptual openness of the simulation tools, and flexible, adaptable and self-learning component libraries will be needed. Associated digital models must be easy to integrate, adaptable, extensible, and easy to reuse in different contexts. For this purpose, they must be modular, easy to parameterize, have flexible structures and, if necessary, be generated automatically (Wenzel et al. 2018).

The focus of this paper is therefore on the presentation of trends in automatic composition of structures for simulation models in production and logistics and the presentation of a developed approach, which enables to generate structures of simulation models from prefabricated component collections in a simulation tool, which reduces the modeling effort. With this approach, the suitable structures of a simulation model should be selectable and adaptable. The paper is structured as follows: After this introduction developments and trends in automatic model generation are briefly discussed in Section 2. Subsequently, possibilities from the field of software engineering are explained, which illustrate a possible approach for the composition of structures for simulation models (Section 3). In particular, the synthesis of logical structures using combinatorial logic is presented. The developed approach, how this logic can support the automated generation of structure variants in simulation models, is shown by means of a simple example. In this example different possibilities of structure variants are clarified and their potential for an automated composition is pointed out (Section 4). Finally, a the results (Section 5) and some information about future developments are discussed.

2 DEVELOPMENTS AND TRENDS IN AUTOMATIC MODEL GENERATION

Lattner et al. (2011) identify different approaches to improve the performance of simulation studies for production and logistics and developed a methodology to automate simulation studies in logistics and to ensure high quality simulation results. Research efforts on *procedure models*, *automatic model generation concepts*, *auxiliary tools for statistics*, *optimization or visualization* and *decision support systems* are presented (Lattner et al. 2011). Research work and concepts for automatic model generation are of particular importance for the paper. Therefore, they will be explained more detail below.

The *automatic model generation concepts* support the execution of a simulation study by using existing data sources to generate simulation models of a specific manufacturing system. In many industries, it is very common for process flow information to be stored electronically, e.g. in the Manufacturing Execution System (MES). This data can be exported to parameterize the existing simulation models or to use this data for semi-automatic or automatic generation of simulation models from a specific area. For the semi-automatic or automatic generation of simulation models with the help of data, suitable technical concepts must be used (Fowler and Rose 2004). Bergmann and Strassburger (2010) call these technical concepts *data-driven model generation*. Semi-automatic data-driven model generation and initialization promise to support the goals and execution of a simulation study consistently and effectively. In general, the term semi-automatic model generation in the simulation context means an approach in which the simulation model generation is not carried out manually with the modeling tools or the programming environment, but rather is generated from external data sources using interfaces and algorithms (Bergmann and Strassburger 2010).

According to Bergmann and Strassburger (2010) and Gmilkowsky et al. (1998), the approaches of automatic data-driven model generation are roughly divided into the following classes. The three relevant classes for the automation of the building of DES models are *parametric*, *structure-based* and *hybrid knowledge-based* approaches.

Parametric approaches are characterized by the fact that models are built on the basis of existing simulation modules stored in libraries, which are selected and configured by the model generator on the basis of parameters (Bergmann and Strassburger 2010). For example, the approach of Skoogh et al. (2012) refers to the parameterization of a model and includes all steps from the preparation of the raw data to their transformation into input data for a simulation model.

The starting point for model generation in *structure-based approaches* is data describing the structure of the system to be mapped, in particular, factory layout data from corresponding CAD systems (Bergmann and Strassburger 2010). These include such approaches as the one approach of Lorenz and Schulze (1995) and the approach of Splanemann (1995). Lorenz and Schulze (1995) use CAD data and data from other information systems for semi-automatic model generation. The approach for the semi-automatic generation of simulation models based on a STEP data exchange format (STEP: Standard for the Exchange of Product model data) by Splanemann (1995) also uses the layout as a primary data source in the form of CAD data.

Hybrid knowledge-based approaches combine methods of artificial intelligence (e.g. expert systems, neural networks) with the two approaches described above (Bergmann and Strassburger 2010). Hybrid approaches include the approaches, e.g. developed by Gmilkowsky et al. (1998) and Selke (2004). The vast majority of current semi-automatic or automatic model generation methods (Bergmann and Strassburger 2010) can be classified as *"hybrid"* because they usually require additional expert knowledge for model generation. Therefore, most of the approaches cannot be clearly classified as parametric approaches or structure-based approaches.

A lot of approaches to semi-automatic and automatic model generation are not universal and are often limited to one question or one of the life cycle phases of a factory (Bergmann and Strassburger 2010). For example, approaches such as Rooks (2009) support the planning phases of systems and approaches such as Splanemann (1995), Selke (2004), Jensen (2007) support the operating phase of a factory.

An essential problem of automatic model generation is the representation of more complex system behavior. The spectrum of problems already starts with the mapping of simple control strategies. Rooks (2009) uses digital process planning as a data source for his developments and supplements this with layout information from another system. The approach presented by Rooks (2009) requires that the typically missing descriptions of rules of conduct be manually supplemented by the planners. This approach technically allows to generate executable simulation models semi-automatically, but it is questionable whether this approach can be implemented in practice. The work of Selke (2004) examines a different approach to the problem of generating control strategies. Here the model generation is located in a companion context and an attempt is made to recognize the control strategies automatically by evaluation of status data of the real system. The automatic generation of such behavior rules and logics is difficult and typically limited to simple decision rules or decision tables. Jensen (2007) develops a method for coupling different industrial production-related data storage systems with the goal of automatic model generation. The core of his work is the merging of different data sources into a data framework based on web services. These contain a logic with which these data can be flexibly processed in an Extensible Markup Language (XML) file. With this approach, questions such as capacity analyses, lot size determination and buffer dimensioning can be addressed.

Mueller et al. (2007) presents a simulation framework that can generate a wide variety of manufacturing simulation models. This approach is based on the description of models using simulation data and aims at reducing the effort for verification and validation of models. The approach according to Mueller et al. (2007) uses existing theory for classical Petri nets and allows to make certain statements about the behavior of the generated simulation model. In Ruscheinski et al. (2018) a methodology for conducting simulation experiments is being developed. This approach supports the automatic generation of simulation experiments and facilitates the planning and execution of simulation experiments. A template for various simulation experiments and the documentation of a simulation model are used to automatically generate and execute statistical model test experiments and sensitivity analysis experiments. In this approach, the documentation of the experiments is also generated semi-automatically by the simulation expert using the documentation during model development to generate simulation experiments.

In summary, it can be stated that various practical approaches exist towards generating simulation models semi-automatically. In most cases, however, these approaches are tailored to a specific operating scenario or research goals. Furthermore, a strict separation into planning and operational approaches can be observed. The technical implementations are very different due to the lack of standards. In addition, the variation of model structures is not taken into account in automatic model generation, for example in order to achieve the best possible system variant with regard to the simulation problem. The procedure for selecting a model with associated structural variants is also only considered statically in the approaches, whereby the future simulation models of the production systems must have the ability to adapt outside the corridor with predefined system solutions. Therefore, the simulation models should not only be generated automatically, but also the adaptability of the structures has to be considered. Approaches that consider adaptability are still rare. In the following section, an approach is presented in which structural variants of simulation models can be generated with the help of combinatory logic.

3 COMBINATORY LOGIC

A collection of several, structurally different, but largely identical contents of a (simulation) model or a software solution is also referred to, in computer science, as a (software) product line. The variability of simulation model structures can be defined as product lines of simulation models and poses particular challenges to software. Each point of variation can be realized by a set of features whose cardinality multiplicatively enters into the number of possible occurrences of the product line. With eight variation points with two possible implementations, up to 256 possible product characteristics are already created in this way. Consequently, it is extremely challenging to create and test the software of a realistic product line without automation and reuse. Furthermore, the multiplicative relationship between the costs for the selection and integration of features asymptotically always exceeds the additive connection between the total costs and the realization of separate features. Combinatory software synthesis is motivated by this cost composition. Independently implemented features in the form of a component repository are assumed and their selection and composition are automated by an algorithm.

The idea of combinatory logic (CL) goes back to basic works for modern mathematics by David Hilbert and Haskell Curry (Curry 1930; Krazer 1905). Questions of type-inhabitation are mathematically well-defined and enjoy a likewise rich research tradition (Statman 1979; Urzyczyn 1999; Düdder et al. 2012; Rehof and Urzyczyn 2011). The decidability and complexity of the restricted combinatory logic used with intersection types has been mathematically proven (Düdder et al. 2012). An implementation of the inhabitation algorithm suitable for the research project exists in the form of the Combinatory Logic Synthesizer (CLS) Framework (Bessai et al. 2014). The application has been successfully tested in numerous experiments (Bessai 2013; Vasileva 2013; Wolf 2013; Düdder et al. 2012). The relationship between type inhabitation and feature diagrams for software product lines has been formally proven mathematically (verified by a theorem prover), confirmed experimentally using a prototypical example. The formal proof and the application are published by peer review (Bessai et al. 2016).

The heterogeneity of sub-problems in creating software makes it indispensable to use a synthetic approach that is strong enough in its expressive power to specify highly specialized issues in a problem-dependent vocabulary. This requirement can be met by mapping to the synthesis problem of combinatory logic (Rehof 2013). Formally, the repository is interpreted as a collection Γ of combinators whose type corresponds to the interface specification of the respective software component. By using the intersection type system, types can be simultaneously extracted from the type system of the programming language used for implementation, and can include additional semantic characterizations. An example for such a semantic characterization is the function "*Cel2Fhrt:* $(int \rightarrow int) \cap (Cel \rightarrow Fhrt)$". Let us assume that the function converts a Celsius temperature into the corresponding Fahrenheit temperature. "Cel2Fhrt" denotes the function name, $(int \rightarrow int)$ the signature of the function with regard to native types. The function accepts an integer number as an argument and returns an integer number as the return value. $(Cel \rightarrow Fhrt)$ adds a semantic description of the function argument and the return value to the type signature. The integer number of the function argument therefore has the semantic meaning of a Celsius temperature, the return value therefore represents a Fahrenheit temperature. Types of this type make it possible to specify components in detail and to ensure that their composition is also correct in semantic terms when answering the type-inhabitation question.

The type-inhabitation question is answered in an algorithmic way: Can be from a given set of components (or combinators) "Γ" create a term "e" with the desired target type "τ" ("$\exists$ e. $\Gamma \vdash e: \tau$")?

In the subsequent code generation, executable program code is generated from the developed models, without this having to be written by a developer "by hand". There are certainly similarities to compilers that also generate machine readable and executable programs from a previously written source code. However, code generators generate the source code that the compiler can accept. Just as there are separate programming languages for formulating source code, there are also languages for formulating models from which source code can be generated.

This method is now to be used in this application to generate executable simulation models from a collection of appropriately structured building blocks. During code generation, executable program code is generated from a automatically composed model without this having to be written "manually" by a developer. There are certainly similarities to compilers that also generate machine readable and executable programs from a previously written source code. However, code generators use one step earlier and generate the source code that the compiler can accept. Just as there are separate programming languages for formulating source code, there are also languages for formulating models from which source code can be generated.

4 APPLICATION AND EXAMPLE

The Combinatory Logic Synthesizer (CLS) (Bessai et al. 2014) uses the principle of template-based code generation. A template describes a program code written as a kind of gap text. While parts of the code look like normal application code (JAVA code in the present case), other parts consist of wildcards that are populated according to requirements during the generation process. In the CLS framework, individual components can be added to the components. If a component is used in a solution, its code block will be inserted at the intended location in the entire template of the solution. At the end of the process, executable program code with the previously specified requirements is created.

It is certainly easy to see that the approach of software synthesis by combinatory logic presented in the previous section is very well suited to automatically generate simulation models as well. This is mainly due to the fact that simulation models are also represented as program code in most of the common tools. For example, the AnyLogic simulation environment generates JAVA code executable from the graph created in the editor, which of course could also be generated by the CLS system. Other simulation environments, such as Plant Simulation, work in the same way, although in some cases with other programming languages. In order to clarify the general feasibility of the project, the general planned procedure will now be explained below with the help of a smaller example (Figure 1).

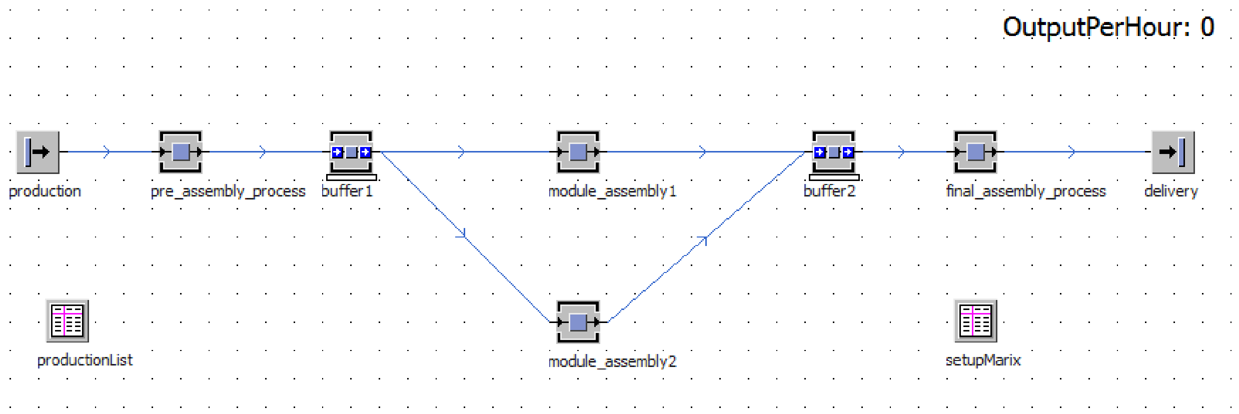

Figure 1: Exemplary simulation model.

Figure 1 shows a simple exemplary simulation model of a production line. In the example, a work piece is manufactured by a pre-assembly and a final assembly process. The aim is to determine the possible throughput per hour. In the model, in the sense of structural variance, the use of buffers in the production line, as well as the decision which modules are used (only module_assembly1, only module_assembly2, or both) should be optional. In order to determine which configuration of the process promises the highest throughput per hour, the simulation expert have to individually model and evaluate each of the 12 resulting opportunities. This is associated with a lot of modeling effort even in a small example. By contrast, the approach planned in this project allows the simulation expert to model only the individual components of the process in an editor yet to be developed, and to provide them with appropriate types that reflect the dependencies in the model. For the example used in this paper, the components are represented in a repository in Figure 2.

```
module_assembly_group2:
(String -> String -> String) & (buffer1 -> module_assembly1 ->
module_assembly_process)

buffer1: (String -> String) & (pre_assembly_process -> buffer1)

buffer2: (String -> String) & (module_assembly_process -> buffer2)

module_assembly1: String & module_assembly1

module_assembly2: String & module_assembly2

module_assembly_group5: (String -> String) & (buffer1 -> module_assembly1
-> module_assembly_process)

module_assembly_group1: (String -> String -> String -> String) & (buffer1 ->
module_assembly1
-> module_assembly2 -> module_assembly_process)

module_assembly_group6: (String -> String -> String) & (pre_assembly_process ->
module_assembly2 -> module_assembly_process)

start: String & start

end: (String -> SimModelCompilationUnit) & (final_assembly -> end)

module_assembly_group4: (String -> String -> String -> String) &
(pre_assembly_process -> module_assembly1 -> module_assembly2 ->
module_assembly_process)

module_assembly_group4: (String -> String -> String) & (pre_assembly_process ->
module_assembly2 -> module_assembly_process)

final_assembly: (String -> String) & (module_assembly_process -> final_assembly)

final_assembly2: (String -> String) & (buffer2 -> final_assembly)

pre_assembly: (String -> String) & (start -> pre_assembly_process)
```

Figure 2: Combinator repository for simulation model synthesis.

The repository is formed from the individual modules of the simulation model shown above. This has been broken down into its components and further variants are to be created from these components. Essential is the combinator called "end". It generates in JAVA code templates, which can later be used in

simulation tools. All other combinators (or components) then add further components of the simulation model to be generated to this template. The dependencies between each combiner are defined by the semantic types in the combiner's signature. Thus, the signature of the combiner "final assembly" (Buffer 2 => final assembly) states that the code of the combiner "buffer2" must be present to generate the code from the combiner "final_assembly". Structural variance is covered by the fact that there is another combiner of the type "final_assembly" in the present repository. "final_assembly" has the signature "module_assembly_process => final_assembly", which thus does not require the buffer block.
The inhabitation algorithm generates in this case all valid variants of the process, so both a solution that uses the first and a solution that uses the second “Final Assembly” combinator. In this case, a corresponding program code is generated in the background, which could be translated into code when developing the corresponding interfaces, which is accepted by a desired simulation software. This automatically creates a whole series of simulation models covering all possible variants that can occur in this model. Some of the variants of the generated structures can be seen in Figure 3.

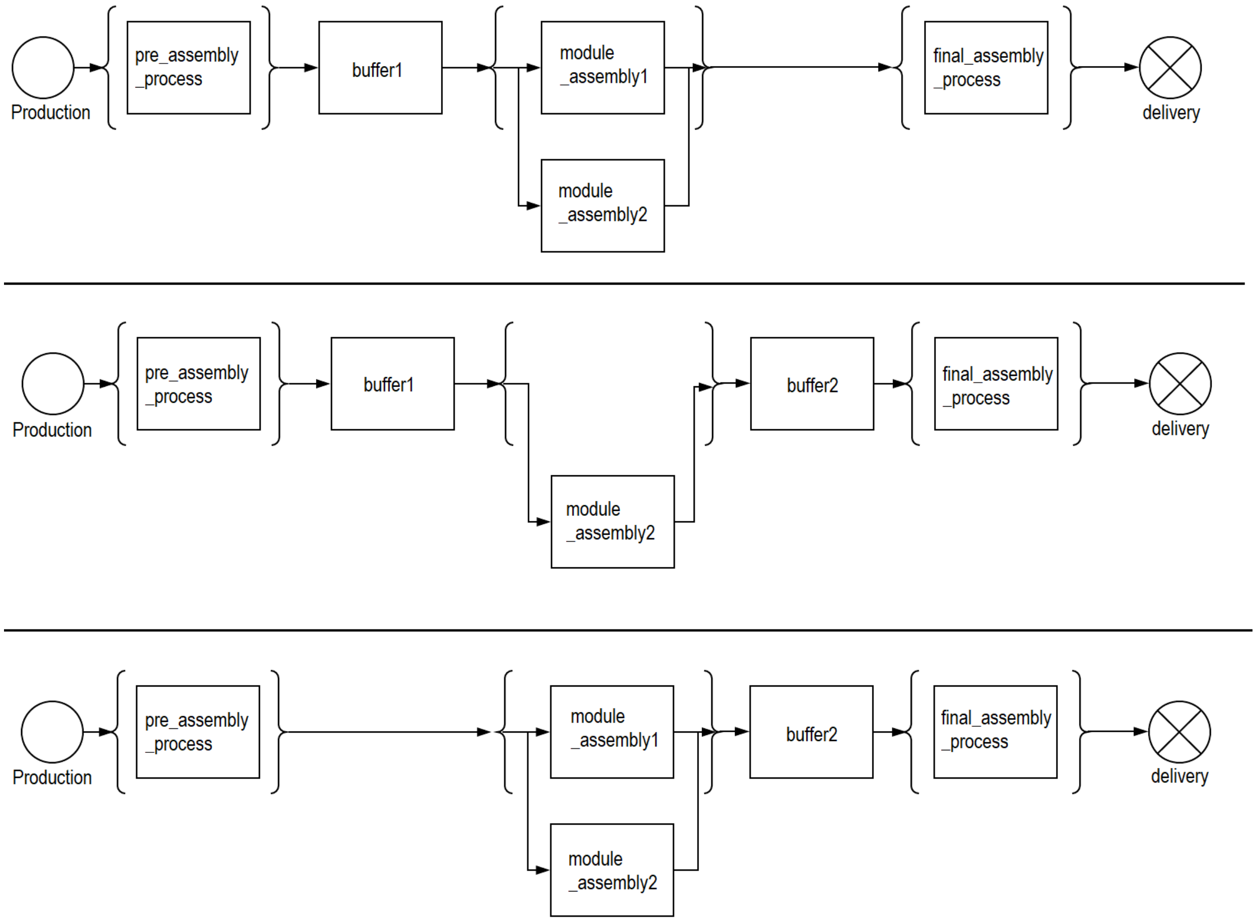

Figure 3: Excerpt from the generated model structures.

Even if only a part of the generated variants can be seen here, it is still ensured that the set of possible variants of the process is generated completely. In addition to the structures, the control logics can also be generated. This can be done by using appropriate combinators, which contain the control code. At the end of the generation process, all variants of a simulation model are available in executable form. These can then be executed and evaluated in the simulation tools, where it can be determined which of the structure variants is the most suitable for the respective application.

5 CONCLUSION AND FUTURE WORK

In this paper, we have reviewed the current state of research in the automated generation of structural variants of simulation models of production systems. Our research has shown that, despite extensive work in the late 1990s, there are still a few executable and resilient approaches. Thus, structural variance still causes a large amount of manual work in the generation of simulation models, which can only be absorbed by complex remodeling of existing models.

Subsequently, we have identified an approach to address this gap. In our approach, structures of simulation models are generated from prefabricated component collections in a simulation tool, thus reducing modeling work. The goal of future research will be to further develop this approach and to implement a framework for the automatic generation of simulation models. Therefore it is necessary to use a large number of components from which such models can be generated. It is also necessary to process and record the relationships between individual components, their mutual relationships, and their associated control logics in a structured way. For this purpose it is our goal to design a taxonomy that maps the entire domain of the simulation components to create a structured semantic basis for the component-based synthesis of structural variants of simulation models.

REFERENCES

Abel, D., S. Wenzel, and Jessen. 2013. “Delphi-Study on evaluating information in simulation studies for manufacturing and logistics planning”. *Journal of Simulation* 7(4):240-248.

Bergmann S. and S. Strassburger. 2010. “Challenges for the automatic generation of simulation models for production systems”. In *Proceedings of the 2010 Summer Simulation Multiconference (SummerSim'10)*, July 11th-15th, Ottawa, Canada, 545-549.

Bessai, J. 2013. “Synthesizing Dependency Injection Configurations for the Spring Framework”. Master Thesis, Dortmund: TU Dortmund University, Department of Computer Science.

Bessai, J., A. Dudenhefner, B. Düdder, M. Martens, and J. Rehof. 2014. “Combinatory Logic Synthesizer”. In *Part I of the Proceedings of the 6th International Symposium on Leveraging applications of Formal Methods, Verification and Validation. Technologies for Mastering Change - Volume 8802*, edited by T. Margaria and B. Steffen, 26-40. New York: Springer-Verlag New York, Inc.

Bessai, J., B. Düdder, Boris, G. T. Heineman, and J. Rehof. 2016. “Combinatory Synthesis of Classes Using Feature Grammars”. In *Formal aspects of component software. 12th international conference, facs,* edited by C. Braga and P. Csaba Ölveczky, 123-140. Rio de Janeiro, Brazil: Springer Lecture Notes in Computer Science.

Bogon, T., U. Jessen, A. Lattner, D. Paraskevopoulos, M. Schmitz, S. Spieckermann, I. J. Timm, and S. Wenzel. 2012. “Towards Assisted Input and Output Data Analysis in Manufacturing Simulation: The EDASim Approach”. In *Proceedings of the 2012 Winter Simulation Conference*, edited by C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, 2806-2818. Berlin, Germany: Institute of Electrical and Electronics Engineers, Systems, Man, and Cybernetics Society (IEEE/SMC).

Bracht, U., D. Geckler, and S. Wenzel. 2018. *Digitale Fabrik – Methoden und Praxisbeispiele*. 2rd ed. Berlin: Springer, Inc.

Curry, H. B. 1930. "Grundlagen der kombinatorischen Logik ". *American journal of mathematics*, 52(4):789-834.

Düdder, B., O. Garbe, M. Martens, J. Rehof, and P. Urzyczyn. 2012. *Using Inhabitation in Bounded Combinatory Logic with Intersection Types for Composition Synthesis*. Technical Reports In Computer Science, Department of Computer Science, Technical University of Dortmund, Dortmund. https://www-seal.cs.tu-dortmund.de/seal/downloads/research/cls/TR842-ITRS.pdf, accessed 12th July 2019.

Fowler, J. W. and O. Rose. 2004. “Grand Challenges in Modeling and Simulation of Complex Manufacturing Systems”. *SIMULATION: The Society for Modeling and Simulation International* 80(9): 469-476.

Gmilkowsky, P., E. Eckardt, and D. Palleduhn. 1998. “Automated simulation model generation: a knowledge based approach within an integrated system for modeling, simulation and analysis of discrete production processes”. In *Proceedings of the Simulators International XV*, edited by R. Ades and M. Griebenow, 157-162. San Diego, California: Society for Computer Simulation.

Hotz, I., A. Hanisch, and T. Schulze. 2006. “Simulation-Based Early Warning Systems as a Practical Approach for the Automotive Industry”. In *Proceedings of the 2006 Winter Simulation Conference*, edited by L. F. Perrone, F. P. Wieland,J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 1962-1970. Monterey, California: Institute of Electrical and Electronics Engineers, Inc.

Jankauskas, L. and S. McLafferty. 1996. "BESTFIT, distribution fitting software by Palisade Corporation". In *Proceedings of the 1996 Winter Simulation Conference*, edited by J. M Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 551-55. Coronado, California: Institute of Electrical and Electronics Engineers, Inc.

Jensen, S. 2007. *Eine Methodik zur teilautomatisierten Generierung von Simulationsmodellen aus Produktionsdatensystemen am Beispiel einer Job Shop Fertigung*. Kassel: University Press.

Krazer, A. 1905 „Über die Grundlagen der Logik und der Arithmetik". In *Verhandlungen des 3. Internationalen Mathematiker-Kongresses in Heidelberg vom 8. bis 13. August 1904,* edited by A. Krazer, 174-185, Leipzig, Heidelberg: University Library.

Lattner, D. A., H. Pitsch, I. J. Timm, S. Spieckermann, and S. Wenzel. 2011. "AssistSim – Towards Automation of Simulation Studies in Logistics". *SNE Simulation Notes Europe* 21(3-4):119-128.

Law, A. M. and M. G. McComas. 1989. "Pitfalls to avoid in the simulation of manufacturing systems". *Industrial Engineering* 31:28-31.

Lorenz, P. and T. Schulze. 1995. "Layout Based Model Generation". In *Proceedings of the 1995 Winter Simulation Conference*, edited by W. R. Lilegdon, D. Goldman, C. Alexopoulos and K. Kang, 728-735. Arlington, VA: Institute of Electrical and Electronics Engineers, Inc.

Meyer T. and S. Straßburger. 2012. "Using protocol state machines to support simulation-based emulation projects". In *Proceedings of the European Simulation and Modelling Conference 2012,* edited by M. Klumpp, 234-238. Essen: EUROSIS-ETI.

Mueller, R., A. Christos, and F. L. McGinnis. 2007. "Automatic generation of simulation models for semiconductor manufacturing". In *Proceedings of the 2007 Winter Simulation Conference,* edited by S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 648-657. New York: Institute of Electrical and Electronics Engineers, Inc.

Ozdemirel, N. E., G. T. Mackulak, and J. K. Cochran. 1993. "A group technology classification and coding scheme for discrete manufacturing simulation models". *International Journal of Production Research* 33(3):579-601.

Rabe, M., S. Spieckermann, and S. Wenzel. 2009. "Verification and Validation Activities within a New Procedure Model for V&V in Production and Logistics Simulation". In *Proceedings of the 2009 Winter Simulation Conference,* edited by M. D. Rossetti, R. R. Hill, B. Johansson, A. Dunkin, and R. G. Ingalls, 2509-2519. Austin, Texas: Institute of Electrical and Electronics Engineers, Inc.

Rehof, J. and P. Urzyczyn. 2011. "Finite Combinatory Logic with Intersection Types". In: *Typed lambda calculi and applications. 10th international conference, TLCA 2011*, edited by L. Ong, 169-183. Berlin, Heidelberg: Springer.

Rehof, J. 2013. *Towards Combinatory Logic Synthesis.* In *BEAT'13: 1st International Workshop on Behavioural Types*. https://www-seal.cs.tu-dortmund.de/seal/downloads/rehof/research_papers/Beat13rehof.pdf, accessed 15th July 2019

Rooks, T. 2009. *Rechnergestützte Simulationsmodellgenerierung zur dynamischen Absicherung der Montagelogistikplanung bei der Fahrzeugneutypplanung im Rahmen der Digitalen Fabrik*. Clausthal: TU Clausthal-Zellerfeld.

Ruscheinski, A., K. Budde, T. Warnke, P. Wilsdorf, B. C. Hiller, M. Dombrowsky, and A. M. Uhrmacher. 2018. "Generating simulation experiments based on model documentations and templates". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 715-726. New York: Institute of Electrical and Electronics Engineers.

Selke, C. 2004. *Entwicklung von Methoden zur automatischen Simulationsmodellgenerierung*. Munich: Technical University of Munich.

Schluse, M. and J. Rossmann. 2016. „From Simulation to Experimentable Digital Twins: Simulation-based Development and Operation of Complex Technical Systems". In *Second IEEE International Symposium on System Engineering (ISSE 2016),* October 3th-5th, Edinburgh, Scotland, 273-278.

Splanemann, R. 1995. *Teilautomatische Generierung von Simulationsmodellen aus systemneutral definierten Unternehmensdaten*. *Bremer Schriften zur Betriebstechnik und Arbeitswissenschaft Band 5.* Bremen: University Bremen.

Statman, R. 1979. "Intuitionistic propositional logic is polynomial-space complete". In *Theoretical Computer Science*, 9(1):67–72.

Tabeling, P. 2006. *Softwaresysteme und ihre Modellierung. Grundlagen, Methoden und Techniken*. Berlin: Springer.

Urzyczyn, P. 1999. "The emptiness problem for intersection types". In *The Journal of Symbolic Logic*, 64(03):1195–1215.

Vasileva, A. 2013. „Synthese von Orchestrationscode für Cloud-basierte Dienste". *Diploma Thesis*. Dortmund: TU Dortmund, Faculty of Computer Science.

Wolf, P. 2013. "Entwicklung eines Adapters mit VI Scripting (LabVIEW) zur Synthese von LEGO® NXT-VIs aus einem Repository". *Bachelor Thesis*. Dortmund: TU Dortmund, Faculty of Computer Science.

Xu, J., E. Huang, L. Hsieh, L. H. Lee, J. Qing-Shan, and C. H. Chen. 2016. "Simulation optimization in the era of Industrial 4.0 and the Industrial Internet". *Journal of Simulation* (10):310-320.

Wenzel, S., P. Boyaci, and U. Jessen. 2010. "Simulation in Production and Logistics: Trends, Solutions and Applications". In *Advanced Manufacturing and Sustainable Logistics. Proceedings of the 8th International Heinz Nixdorf Symposium,* edited by W. Dangelmaier, A. Blecken, R. Delius, and S. Klöpfer, 73-84. Berlin, Heidelberg: Springer.

Wenzel, S., J. Stolipin, and U. Jessen. 2018. „Ablaufsimulation in Industrie 4.0: Handlungsfelder für die industrielle digitale Transformation". *Industrie 4.0 Management* 34(3):29-32.

AUTHOR BIOGRAPHIES

SIGRID WENZEL is a Professor and head of the Department of Production Organization and Factory Planning, University of Kassel. In addition to this, she is a board director of the Arbeitsgemeinschaft Simulation (ASIM), spokesperson for the ASIM Section Simulation in Production and Logistics, member of the advisory board of the Association of German Engineers Society of Production and Logistics (VDI-GPL), and head of the Committee Modeling and Simulation of the VDI-GPL. Her email adress is s.wenzel@uni-kassel.de.

JAKOB REHOF holds a joint position as full professor of Computer Science at the University of Dortmund, where he is chair of Software Engineering, and as director of the Fraunhofer Institute for Software and Systems Engineering (ISST) Dortmund. Jakob Rehof studied Computer Science and Mathematics at the University of Copenhagen and got his Ph.D. in Computer Science at DIKU, Department of Computer Science, University of Copenhagen. His main field of research is the automatic synthesis of programs from component collections using combinatory logic. His email adress is jakob.rehof@tu-dortmund.de.

JANA STOLIPIN is a research assistant and PhD student at the Department of Production and Factory Planning at the University of Kassel, Germany. Her research focuses on material flow in production and logistics and on knowledge management in simulation studies. Within this research area she investigates the reusability of simulation knowledge in the context of simulation-based projects in intralogistics. Her email adress is jana.stolipin@uni-kassel.de.

JAN WINKELS is a research assistant and PhD student at the Department of Computer Science at the TU Dortmund University in Dortmund, Germany. He works at the Chair for Software Engineering with his research focus on automation of planning processes. Within the scope of this research area he investigates the applicability of the automatic software synthesis in relation to the automation of planning of factory systems. His email adress is jan.winkels@tu-dortmund.de.