Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 104 (2021) 1440–1445

www.elsevier.com/locate/procedia

54th CIRP Conference on Manufacturing Systems

Automatic Building of a Repository for Component-based Synthesis of Warehouse Simulation Models

Fadil Kallat[a,*], Jakob Pfrommer[b], Jan Bessai[a], Jakob Rehof[a], Anne Meyer[c]

[a]*Software-Engineering by Algorithms and Logic, Department of Computer Science, TU Dortmund University, Otto-Hahn-Straße 12, 44227 Dortmund, Germany*
[b]*Production Management and Factory Planning, Faculty of Mechanical Engineering, TU Dortmund University, Leonhard-Euler-Str. 5, 44227 Dortmund, Germany*
[c]*Digitalization in Logistics and Supply Chain Management, Faculty of Mechanical Engineering, TU Dortmund University, Leonhard-Euler-Str. 5, 44227 Dortmund, Germany*

Abstract

Simulations are a common tool in the warehouse planning and adoption process for evaluating and comparing variants of a storage system. But simulation modeling is a complex and time-consuming task. Due to limited resources, often not all possible system variants can be modeled. A promising solution is the migration of an existing simulation model to enable component-based software synthesis. An inhabitation algorithm composes structural variants according to a synthesis goal given a repository of typed components. In this paper, we automatically generate a repository and synthesize simulation model variants using a block stacking warehouse simulation model as an example.

Keywords: storage planning and adoption; block stacking warehouse; automatic simulation model generation; component-based software synthesis

1. Introduction

In factory and warehouse planning a wide range of decisions must be made. Often the decision-making process is supported by simulation studies. But creating simulation models is a time-consuming and costly process, in particular if a large number of variants can be considered. Due to budget and time restrictions, planners may commit on chosen variants, possibly leaving out variants which would have offered a better solution.

In the last decades, there has been an increasing amount of published research in the field of the Automatic Simulation Model Generation (ASMG) [1]. Although many approaches allow to generate simulation models successfully, it is still challenging to generate structure variants [2]. Furthermore, many of the existing approaches are tailored to specific use cases [1, 2].

In this paper, we use the component-based software synthesis framework CLS to migrate an existing simulation model into a software product line. Each member of the software product line is a simulation model, which varies in its structure and parametrization. The main novel contribution is that, in contrast to prior work on simulation model generation with CLS [3], software product line members not only vary in their structure, but also are built from components that are automatically extracted from an existing simulation model. We add these components to a repository, which is used for component-based synthesis. In our approach the human input is an important factor. The planner marks variation points by perfoming adjustments to the components in the repository. Then, the simulation model variants are automatically generated with regard to the planners variation points. Our approach enables using component-based software synthesis for domain-experts in logistics without requiring too much expertise in computer science and programming. Further, the automation speeds up the migration process from individual simulations to software product lines. Our approach supports arbitrary simulation models that are composed from elements of the AnyLogic 8 Process Modeling and the Material Handling Library.

We demonstrate our approach by migrating a block stacking warehouse simulation model into a software product line. We vary the number of in- and outbound docks (I/O-points), the storage strategy and relocation capability. Then, the synthesized block stacking warehouse variants are evaluated to find the most promising configurations.

[*] Corresponding author. E-mail address: fadil.kallat@tu-dortmund.de

This paper is structured as follows: Section 2 presents related work on the component-oriented automatic generation of simulation models and introduces the composition synthesis framework CLS. Section 3 discusses the general process flow and variability points in a block stacking warehouse. Automatic component extraction and simulation model synthesis is presented in Section 4. Section 5 contains an experimental evaluation. Results and future work are discussed in Section 6.

2. Related Work

2.1. Automatic Simulation Model Generation

Automatic Simulation Model Generation (ASMG) is a common tool to support decision-making in factory and warehouse planning. A recent review of the literature on this topic has been conducted by Reinhardt et al. in [1]. The authors focused on the adaptivity of production systems processes. Mourtzis [4] highlights that manual modeling of material flow simulations is a very time-consuming and inefficient process to the point where realistic models are very difficult to realize with existing techniques. The effort presented in this paper aims to close this gap by automating model construction in novel ways.

Several approaches in ASMG investigate the usage of component libraries. Lee and Zobel present a representation methodology to store components and to generate simulation models [5]. The authors use an object-oriented database to generate simulation models by querying the corresponding components. They wrap the components into objects and attach additional information, thereby solving problems of model incompatibility, coupling different simulation techniques, and enhancing the reusability of components.

Verbraeck and Valentin [6] developed design guidelines with regard to simulation components. The authors establish the reusability of similiar components in different simulation projects by having a library of components.

Röhl and Morgenstern [7] combine the Web Service Description Language with XML Schema Definitions to describe model component interfaces. Besides required input and output ports of components, they also consider semantic information. The authors emphasize that their component architecture allows decentralized simulation model development using information about interfaces. Component interface information in form of type specifications is also the central tool of the synthesis framework component library developed here.

Neyrinck et al. [8] use a library of company-neutral and parameterizable basic components in the domain of mechatronics to generate simulation model variants. Beside the components, they introduce skills which are constraints that influence the parameterization of the variants during the generating process. Their approach allows users without simulation modeling knowledge to find promising module configurations and to generate the simulation model. We also address the accessibility for users without advanced knowledge in programming. In contrast to existing work, we focus on structural variance which previous approaches, to our knowledge, have not considered.

2.2. Combinatory Logic Synthesizer

In our approach product line members are generated by the Combinatory Logic Synthesizer Framework (CLS) [9]. It operates with a repository of components, called combinators, which have their inputs and outputs specified by types. Components are combined using the simple but very expressive rules of finite combinatory logic with intersection types [10]. The main driver of composition is the modus ponens rule "if $M : A \to B$ and $N : A$ then $MN : B$", which means that any M with a type that takes A as an input and produces B can be applied to an N of type A to get B. The CLS Framework performs a backwards search starting from the target type (synthesis goal) B to find well-typed compositions of its input components. It additionally employs intersection types $M : A \cap B$ to express that M has two types simultaneously. This is usefull to refine specifications with *semantic types* as in $\text{coldStoreSim} : \text{BufferArea} \cap \text{Refrigerated} \to \text{Output} \cap \text{RefrigeratorTruckCompatible} \to \text{Process} \cap \text{FrozenGoodsCompatible}$, where the additional types guarantee that a warehouse simulation process is built from two components that can handle frozen goods. Compositions MN are automatically translated to function calls $M(N)$ which will generate source code, e.g. for an AnyLogic 8 simulation model. The full algorithmic and logical details as well as the ten years of implementation history of CLS are beyond the scope of this paper, but can be found in [11].

Existing applications of CLS include the synthesis of software product lines [12], workflow plans [13], motion plans [14], and BPMN 2.0 processes [9]. Recent developments have proven CLS to be a useful tool for logistics and factory planning scenarios. In [13] it is used for planning factories, which is often a logistical problem in the broader sense. Starting from the modeling of clinical pathways [15], logistical problems in a more narrow sense have become a focus of CLS applications. The motivation for this, and especially for considering simulation models with structural variability, is discussed in [2]. An example application is the synthesis of simulation model variants for a real-world sheet metal box production line [3] where components were manually designed by a computer science expert. Here, following ideas from [16], a further step toward practical applicability is taken, where logistics experts are enabled to use CLS with automatically extracted component collections. In addition, we enhanced our approach to support migrating various simulation models into software product lines. Further, our approach can now be used to synthesize not only process flowcharts but also visualization and control strategies (user-defined Java functions) of a simulation model.

3. Case Study: Block stacking warehouse

Block stacking warehouses are a very simple type of warehouse and do not require any infrastructure. Unit-loads (e.g. pallets) are placed on a floor and are stacked on top of each other. Even though it seems to be a simple setup, several decision problems with a huge degree of freedom have to be considered in warehouse operations. In autonomously organized ware-

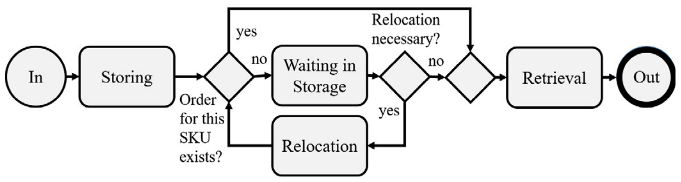


Fig. 1: General process flow of a block stacking warehouse

house, Automated Guided vehicles (AGVs) carry out material handling. The autonomous block stacking warehouse problem proposed by Pfrommer and Meyer [17] gives an overview of all relevant subproblems including the internal layout design problem, the storage location assignment problem (SLAP), the vehicle dispatching problem, the unit-load selection problem and the unit-load relocation problem. The general process flow of the block stacking warehouse is shown in Fig. 1. Unit-loads arrive at an I-point, where they wait to be picked up and be stored in the warehouse. At the placed storage location, the unit-load waits until a retrieval order has to be fulfilled. If there exists already a retrieval order for the Stock Keeping Unit (SKU) at the time of arrival, the unit-load is directly retrieved and delivered to an O-point. Relocation offers the possibility to change the position of unit-loads in order to avoid blocking of other SKUs.

Based on this process flow, an event-discrete simulation model has been developed in Anylogic 8. It has built-in AGV routing which is part of the Material Handling Library. The latter, together with the Process Modeling Library, enables modelling end-to-end processes of a factory or a storage system, thereby speeding up model development. Our model allows us to flexibly adjust and scale the layout, including several storage strategies like Closest Open Location (COL), a random distribution storage strategy, or turnover-based zones to solve the SLAP. It has the possibility to choose between unit-load selection options like Last-In First-Out (LIFO) or First-In First-Out (FIFO) and triggers the relocation of unit-loads if necessary.

4. Synthesis of Simulation Models

In this section, we introduce our approach to automatically extract a repository for component-based synthesis given an AnyLogic 8 simulation model. We concentrate on process-oriented simulation models that use object-oriented flowcharts for specification. The flowcharts are composed from a limited number of parameterizable elements that are part of the Process Modeling and Material Handling libraries. The elements can be enhanced by Java source code that is executed at specific times during the simulation. As result of the object-oriented flowcharts, the simulation models are well-suited for component-based software synthesis. In our work, we synthesize structural variants of the simulation model and their corresponding elements. Fig. 2 shows the workflow and the architecture of our implementation. In our approach, the user uploads the simulation project file to the backend (Step 1). After the generation of the repository of components and the synthesis goal, the user can modify the repository and start the synthesis (Step 2). After synthesis, the user can download the simulation model variant and copy it into a directory that contains the project files (Step 3). The simulation model can directly be executed in AnyLogic 8.

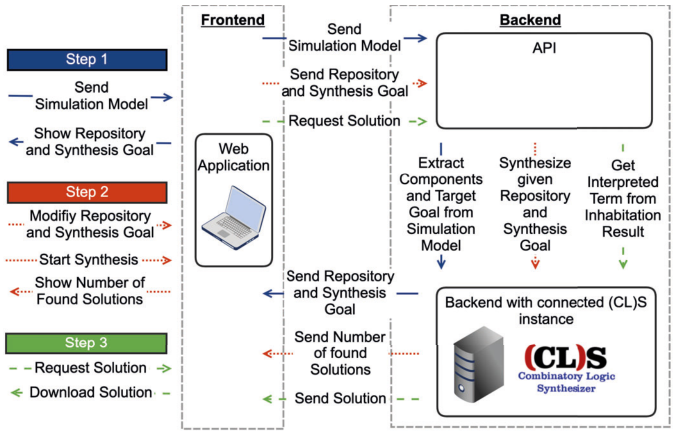

Step 1
Send Simulation Model
Show Repository and Synthesis Goal
Step 2
Modifiy Repository and Synthesis Goal
Start Synthesis
Show Number of Found Solutions
Step 3
Request Solution
Download Solution
Frontend
Web Application
Send Simulation Model
Send Repository and Synthesis Goal
Request Solution
Send Repository and Synthesis Goal
Send Number of found Solutions
Send Solution
Backend
API
Extract Components and Target Goal from Simulation Model
Synthesize given Repository and Synthesis Goal
Get Interpreted Term from Inhabitation Result
Backend with connected (CL)S instance
(CL)S Combinatory Logic Synthesizer

Fig. 2: Workflow and architecture of the implementation. Filled blue lines indicate the first upload step, dotted red lines the second model adjustment step and dashed green lines the final download step.

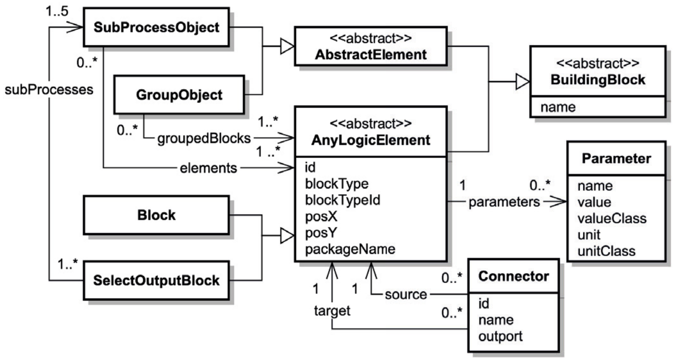


Fig. 3: Data model to represent AnyLogic simulation models. All building blocks of a simulation are classifed according to their function, which is either abstract structuring or performing an elemental computational function.

4.1. Creation of Variability by Adjusting the Repository

The simulation tool AnyLogic 8 stores the simulation models as Extensible Markup Language (XML) files. Their structure allows to automatically extract used elements and the connections between them. Moreover, we extract the parameters of the elements and also global variables, parameters and user-defined Java functions. We transfer the extracted information in a data model that is shown in Fig. 3. The developed data model allows us to represent AnyLogic 8 simulation models.

We automatically create a number of combinators based on the developed data model. Therefore we developed a library of predefined combinator objects. These combinators produce either a single element or a composition of elements. The combinator that produces a single element of the simulation model does not take any arguments. The semantic type of this combinator is an intersection of the capitalized element name and the corresponding type in AnyLogic. By using the unique element name, we achieve a conclusive identification of this combinator.

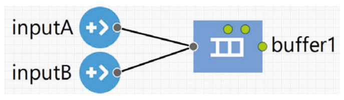


Fig. 4: An AnyLogic process flowchart that consists of two Source blocks (inputA and inputB), which are connected to a Queue block (buffer1).

Further, we predefined three combinators that produce compositions of elements. These combinators take arguments. They produce either a flowchart, a decision element including the connected elements or a group of elements that is characterized by a common successor and/or predeccesor. The semantic type of these combinators is a sequence of semantic types that is seperated by →. The last intersection type in this sequence is the type of the combinator itself. The remaining types are the arguments of the combinator. We assume that the order of the arguments is the same as in the simulation model. We automatically select and add some of the combinators to a repository that is initially empty. We start the synthesis with the goal to synthesize the flowchart. If we do not change any combinators in the repository, then the synthesis result is a simulation model which is the same as the original one.

We enforce the synthesis of variants by adjusting the combinators in the repository. The user can copy or delete a combinator. Moreover, the user can modify the semantic type of an existing combinator to add, replace or remove arguments of this combinator. We implemented a web frontend which allows to perform the adjustments in an accessible manner.

We demonstrate our approach by migrating a simple process flowchart into a software product line. Fig. 4 shows the AnyLogic 8 process which we map into typed combinators. First, we create a number of combinators without arguments for each single element i.e., inputA, inputB and buffer1. Then, we create a group combinator since the elements inputA and inputB are connected to the same successor. Lastly, we create a combinator with arguments that produces the flowchart. This results in the following combinators, which we add to an empty repository:

- inputA : InputA ∩ Source
- inputB : InputB ∩ Source
- buffer1 : Buffer1 ∩ Queue
- groupOfInputs : InputA ∩ Source → InputB ∩ Source → GroupOfInputs ∩ Group
- process : Group ∩ GroupOfInputs → Buffer1 ∩ Queue → Process ∩ Simulation

Given the combinator and the synthesis goal Process ∩ Simulation, the inhabitation algorithm will find a single solution which is equal to the original process flowchart. In the next step, we demonstrate adjustments regarding the components that can be performed by a user. We copy the buffer1 combinator, modify its parameters and rename it to buffer2. Then, we adjust the semantic types of buffer1 and buffer2 to express that they are selectable. Finally, we adjust the semantic type of the combinator that produces the flowchart. The adjustments result in the following combinators that replace the old ones:

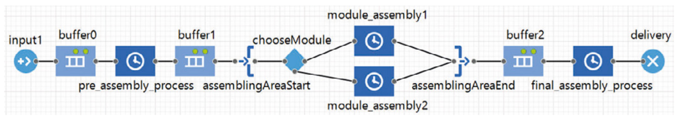


Fig. 5: A simple production line with a pre and final assembly step, two assembly modules, and three buffers modeled in AnyLogic.

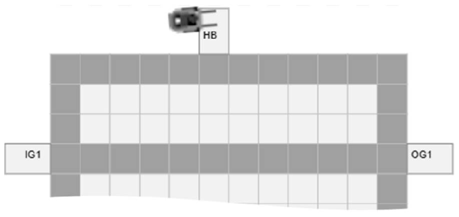


Fig. 6: Excerpt of the grid-based block stacking warehouse scenario with three I- and O-points for inbound (IG) and outbound goods (OG) as well as a hombase (HB) for AGVs.

- buffer1 : Buffer ∩ Queue
- buffer2 : Buffer ∩ Queue
- process : Group ∩ GroupOfInputs → Buffer ∩ Queue → Process ∩ Simulation

Given the adjusted combinators and the previous synthesis goal, the inhabitation algorithm will find two solutions.

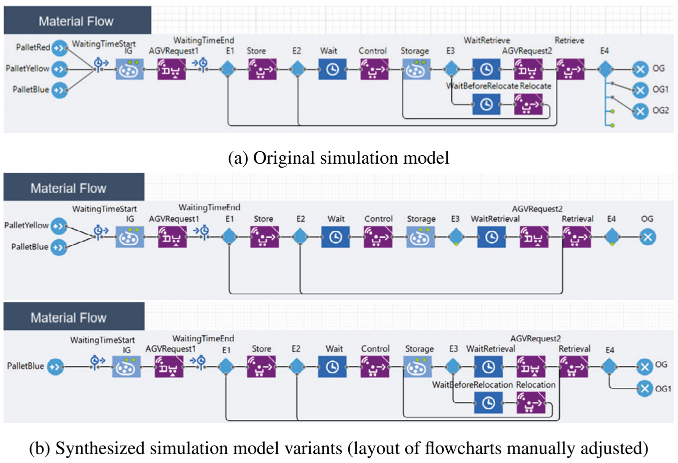
5. Experiments

5.1. *Preliminary experiments*

We performed two preliminary experiments as preparation for our real-world example. We started with a simulation model of a simple production line which consists of a number of queues and assembly steps. Fig. 5 shows the simulation model. In this experiment, we vary the number of assembly modules (1 to 5) and the input blocks (1 to 3). Moreover, we make the final assembly step optional. We used our approach to automatically generate a repository for component-based synthesis. Then, we marked the variation points by adjusting the repository and the synthesis goal in the web application. After starting the synthesis, $5 \times 3 \times 2 = 30$ simulation model variants were synthesized. In our second preliminary experiment, we increased the size and complexity of the simulation model. We migrated a prefabricated simulation model of a job shop that is publicaly available in the AnyLogic Cloud (`https://cloud.anylogic.com/`). In this case, too, our approach automatically generated a repository and we were able to synthesize simulation model variants.

5.2. *Block stacking warhehouse experiment*

Our main experiment is based on a real simulation model of a block stacking warehouse. We consider useful variations such as a variable number of I- and O-points, three different storage strategies and the option of relocation capability. Due to limited space, we show only exemplary evaluation results.



(a) Original simulation model

(b) Synthesized simulation model variants (layout of flowcharts manually adjusted)

Fig. 7: Original simulation model flowchart and two synthesized variants.

No. of I-points	No. of O-points	Storage strategy	Relocation
1, 2, 3	1, 2, 3	ABC, Random, COL	y (yes), n (no)

Table 1: Variation points allowing for 52 combinations. Names are abbreviated.

The grid-based layout consists of five storage bays, which can be accessed from six aisles connected via two cross-aisles at the very beginning and end of the warehouse (see Fig. 6). Each storage location of a bay is able to contain two unit-loads stacked on top of each other and can be accessed directly. In case relocation capability is enabled, these stacked unit-loads can be different Stock Keeping Units (SKUs). The number of I-points varies from one to three. Each I-point links to a separate flow of products with overall 18 different SKUs. The first feeds in eight SKUs, the second four SKUs and the third six SKUs. The number of O-points also varies from one to three. Outbound orders for each O-point are generated randomly. Furthermore, three different storage strategies Random, COL and a turnover-based storage strategy (ABC zones) can be selected. The option of relocation capability allows AGVs to relocate a unit-load if access to a requested unit-load is blocked. Even though the number of AGVs can be easily adjusted, we decided to run the experiments with a single AGV. Multiple AGVs would cause frequent deadlocks, because the layout has an aisle-width of one with bidirectional travel. Table 1 gives an overview of all variations. Based on these configuration possibilities, we automatically generated 52 simulation models via the CLS framework. The complete process of generating the variants is demonstrated in the following screencast (`https://doi.org/10.5281/zenodo.4439750`). Fig. 7 shows the original simulation model and two synthesized variants of the block stacking warehouse.

Subsequently, all generated simulation models have been executed manually with a virtual runtime of eight hours each. Running all simulation models, we encountered problems with the ABC storage strategy caused by the overload of storage zones. In our current simulation model, the storage zones are fixed. However, a changing number of SKUs requires to adjust these zones accordingly.

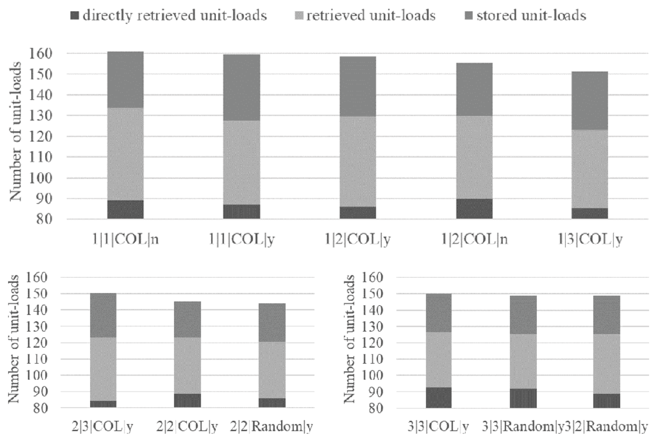


Fig. 8: Warehouse throughput for eight hours of runtime – Overall five best variants as well as below the best three variants for two and three I-points (Abbreviations see Table 1).

Fig. 8 shows the warehouse throughput for eight hours runtime of the best variants comprising stored unit-loads, retrieved unit-loads as well as directly retrieved unit-loads. The five best variants are with one I-point and the COL storage strategy. Due to a small number of SKUs (eight SKUs) for the variants with one I-point, relocation is not helpful. Also, a higher number of O-points is not beneficial, because the requested SKUs can be retrieved quickly. A second and third O-point leads to a longer travel distance and higher staging time of unit-loads. Due to a higher number of SKUs for two and three I-points, relocation capability and a higher number of O-points are beneficial.

Generally, we would expect a higher warehouse throughput for the variants with more I- and O-points. However, the bottleneck in this scenario is the single AGV, which is already at maximum capacity for one I- and O-point. The AGV is travelling 44% of the time loaded with a unit-load as well as 44% of the time empty to pick-up the next one. Loading and unloading of unit-loads requires 12% of the time whereas travelling back to the homebase and idle time account for below 0.15%. This increases mean wait times of unit-loads at the I-points

In summary, the results of our experiment are not surprising. They show that the COL strategy performs better than a random storage strategy. A single I- and O-point are sufficient, since the workload of the AGV is the bottleneck. Relocation of unit-loads becomes interesting for multiple I- and O-points with an increasing number of SKUs. Our issue with the ABC storage strategy shows that some variations have an impact on several parts of the simulation models. This requires many adaptions and is thus easier for an object-oriented structure. In some cases the adaptions require just small changes whereas in other cases a revision of user-defined functions is required. This leads to a situation where planners have to decide if parts of a simulation model should be revised or a number of variants should be omitted. In our approach, we overcome a number of these required adaptions by the use of component-based software synthesis in an additional step for generating user-defined Java functions.

In our work, we document all variants that are not a promising solution. During later stages of the simulation study, the document can be used to understand decisions against some of the rejected variants.

Besides the variations in this experiment and further storage strategies, especially adjusting the number of AGVs and the layout is interesting. Both, the internal and external layout as well as the number of AGVs have a significant impact on the warehouse performance and are in our experiment a limiting factor. The aisle configuration does not allow us to use multiple AGVs and the workload of a single AGV is at maximum capacity. Further addition of battery management of AGVs, a variety of load carriers as well as multiple types of vehicles will allow scenarios that are even more realistic. These additions would undoubtedly lead to more possible variants, which have to be evaluated. Hence, besides the simulation model generation process also the evaluation process needs to be automated.

6. Conclusion

We have developed an approach for migrating existing AnyLogic 8 warehouse and manufacturing simulation models into software product lines. We automated the process of extracting components from a simulation model and building up a repository for component-based synthesis based on combinatory logic with intersection types. Our approach is implemented as a web application that allows to upload simulation models and to synthesize simulation model variants without much expertise in programming and computer science. In this paper, we demonstrated our approach by migrating three different simulation models into product lines. In our main experiment, we synthesized a number of structural variants of a block stacking warehouse simulation model. We investigated the effects of a different number of I- and O-points, various storage strategies and relocation by executing and evaluating their simulations.

To further our research we plan to integrate cloud simulation solutions that allow automatically executing the synthesized simulation model variants. With larger numbers of generated variants, the effort of starting and evaluating the simulation models grows. Further, this automation could be used to automatically rate solutions. This could be helpful since we are currently in the process of investigating the usage of machine learning techniques for the synthesis process. For instance, the choice between a number of selectable components during synthesis could be affected through machine learning.

Moreover, we are planning to integrate the CLS-IDE which allows to investigate synthesis results in our approach. In order to reduce required adaptions to the simulation model, future work could explore modelling guidelines to assure a simulation model that is suitable for component-based synthesis. Simplifying the typing process and integrating a layout engine for synthesized simulation models is also future work.

Acknowledgements

We are grateful for the support by the German Research Foundation (DFG) within the Research Training Group GRK 2193 (www.grk2193.tu-dortmund.de) located in Dortmund.

References

[1] H. Reinhardt, M. Weber, M. Putz, A Survey on Automatic Model Generation for Material Flow Simulation in Discrete Manufacturing, in: Procedia CIRP, Vol. 81 of CIRP, 2019, pp. 121–126. doi:10.1016/j.procir.2019.03.022.

[2] S. Wenzel, J. Stolipin, J. Rehof, J. Winkels, Trends in Automatic Composition of Structures for Simulation Models in Production and Logistics, in: WSC, National Harbor, MD, USA, IEEE, 2019, pp. 2190–2200. doi:10.1109/WSC40007.2019.9004959.

[3] F. Kallat, C. Mieth, J. Rehof, A. Meyer, Using Component-based Software Synthesis and Constraint Solving to generate Sets of Manufacturing Simulation Models, in: Procedia CIRP, Vol. 93 of CIRP, 2020, pp. 556–561. doi:10.1016/j.procir.2020.03.018.

[4] D. Mourtzis, Simulation in the design and operation of manufacturing systems: State of the art and new trends, International Journal of Production Research 58 (7). doi:10.1080/00207543.2019.1636321.

[5] C. Lee, R. Zobel, Representation of Simulation Model Components for Model Generation and a Model Library, in: ANSS, 1996, pp. 193–201. doi:10.1109/SIMSYM.1996.492167.

[6] A. Verbraeck, E. Valentin, Design Guidelines for Simulation Building Blocks, in: WSC, Miami, Florida, USA, WSC, 2008, pp. 923–932. doi:10.1109/WSC.2008.4736158.

[7] M. Röhl, S. Morgenstern, Composing Simulation Models using Interface Definitions based on Web Service Descriptions, in: WSC, Washington, DC, USA, WSC, 2007, pp. 815–822. doi:10.1109/WSC.2007.4419677.

[8] A. Neyrinck, A. Lechler, A. Verl, Automatic Variant Configuration and Generation of Simulation Models for Comparison of Plant and Machinery Variants, in: Procedia CIRP, Vol. 29 of CIRP, 2015, pp. 62–67. doi:10.1016/j.procir.2015.02.069.

[9] J. Bessai, A. Dudenhefner, B. Düdder, M. Martens, J. Rehof, Combinatory Process Synthesis, in: ISoLA, Imperial, Corfu, Greece, Vol. 9952 of LNCS, 2016, pp. 266–281. doi:10.1007/978-3-319-47166-2_19.

[10] J. Rehof, P. Urzyczyn, Finite combinatory logic with intersection types, in: TLCA, Novi Sad, Serbia, Vol. 6690, Springer, 2011, pp. 169–183. doi:10.1007/978-3-642-21691-6/_15.

[11] J. Bessai, A Type-theoretic Framework for Software Component Synthesis, Ph.D. thesis, Technical University of Dortmund, Germany (2019). doi:10.17877/DE290R-20320.

[12] G. T. Heineman, J. Bessai, B. Düdder, J. Rehof, A Long and Winding Road Towards Modular Synthesis, in: ISoLA, Imperial, Corfu, Greece, Vol. 9952 of LNCS, 2016, pp. 303–317. doi:10.1007/978-3-319-47166-2_21.

[13] J. Winkels, J. Graefenstein, T. Schäfer, D. Scholz, J. Rehof, M. Henke, Automatic Composition of Rough Solution Possibilities in the Target Planning of Factory Planning Projects by Means of Combinatory Logic, in: ISoLA, Limassol, Cyprus, Vol. 11247 of LNCS, Springer, 2018, pp. 487–503. doi:10.1007/978-3-030-03427-6_36.

[14] F. Kallat, T. Schäfer, A. Vasileva, CLS-SMT: Bringing Together Combinatory Logic Synthesis and Satisfiability Modulo Theories, in: PxTP, Natal, Brazil, Vol. 301 of EPTCS, 2019, pp. 51–65. doi:10.4204/EPTCS.301.7.

[15] T. Schäfer, F. Möller, A. Burmann, Y. Pikus, N. Weißenberg, M. Hintze, J. Rehof, A Methodology for Combinatory Process Synthesis: Process Variability in Clinical Pathways, in: ISoLA, Limassol, Cyprus, Vol. 11247 of LNCS, Springer, 2018, pp. 472–486. doi:10.1007/978-3-030-03427-6_35.

[16] J. Bessai, M. Roidl, A. Vasileva, Experience Report: Towards Moving Things with Types - Helping Logistics Domain Experts to Control Cyber-Physical Systems with Type-Based Synthesis, in: F-IDE@FM, Porto, Portugal, Vol. 310 of EPTCS, 2019, pp. 1–6. doi:10.4204/EPTCS.310.1.

[17] J. Pfrommer, A. Meyer, Autonomously organized block stacking warehouses: A review of decision problems and major challenges, Logistics Journal: Proceedings 2020 (12). doi:10.2195/lj_Proc_pfrommer_en_202012_01.