Proceedings of the 2022 Winter Simulation Conference
B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds.

AUTOMATIC COMPONENT-BASED SYNTHESIS OF USER-CONFIGURED MANUFACTURING SIMULATION MODELS

Alexander Mages
Carina Mieth

TRUMPF Werkzeugmaschinen SE & Co. KG
Johann-Maus-Straße 2
Ditzingen, 71254, GERMANY

Jens Hetzler

ITK Engineering GmbH
Im Speyerer Tal 6
Rülzheim, 76761, GERMANY

Fadil Kallat
Jakob Rehof
Christian Riest
Tristan Schäfer

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

ABSTRACT

Using simulation models for manufacturing facilities is a common approach for planning, optimizing, and testing different machine configurations and positioning before the actual construction. However, creating these models is time-consuming and costly. Consequently, only a few different simulation models are usually created based on best practices and experience, precluding any examination of the entire variety of possible solutions. To address these obstacles, we present a proof of concept to automate and hence reduce the cost of the process of simulation model creation, thereby allowing for the creation of a larger number of selectable solution variants. Based on a given master simulation model, which obtained all possible variations of a shop floor, we defined simulation building blocks as components. We used component-based synthesis using combinatory logic to synthesize a product line of varying simulation models for a given configuration to be executed and evaluated to find suitable solutions.

1 INTRODUCTION

Simulation of complex processes in manufacturing facilities is a powerful method for production planning and optimization regarding dimensioning machines, production lines, and virtual commissioning (Reinhardt et al. 2019). It allows the evaluation of different variants to support the decision-making process (Negahban and Smith 2014). Nevertheless, setting up and executing a simulation model is often a manual process that involves several steps and experts — for instance, defining simulation scenarios by a production planner, consolidating and preparing input data for the simulation by a data analyst, and implementing simulation models by a simulation expert (Chance et al. 1996). Because of the different process steps and experts, the development cycles of simulation models tend to need time from several days to weeks (Lugaresi

2021). A possible strategy to reduce this effort from days/weeks to hours or minutes is automating the generation of the simulation models for a given scenario and input data. Furthermore, various variants of a manufacturing facility often need to be evaluated in a simulation study to make an informed decision, which leads to adapting the simulation model's parameters, control strategies, and structures (Wenzel et al. 2019). In addition to the time and cost of constructing these simulation models, comparing a large number of variants binds resources and workforce.

To address especially the challenge of generating multiple variants, we present in this paper a proof of concept (PoC) for the automated generation of a product line of simulation models for shop floor layouts of machines in a fabric hall.

Instead of creating new simulation models from scratch, we start with a master simulation model, which allows obtaining all possible variations of a shop floor. However, each variation of the shop floor has to be modeled individually by a simulation expert. We use this master simulation model to define and extract different components, connections, and variation points in the model. A component-based synthesis framework then uses these elements to construct various simulation models depending on a given configuration. In effect, an existing model is thereby migrated to an automated product line of model variants.

Figure 1 gives an overview of the architecture and the workflow within our PoC. The existing master simulation model uses elements from a Model Library (1) developed by TRUMPF and ITK Engineering to represent parametrized machines and their automation units. These elements allow a simulation expert to build a simulation model in a simulation tool. We use the Model Library to implement predefined machine models, which we refer to as machine cells (2). These machine cells form an abstraction layer. Then, we wrap the machine cells into components (3) for component-based software synthesis. Additional components (4) provide the functionality to collect data, such as references to agents or the presentation layer from the original simulation. These references enable further components to modify the simulation model, such as producing paths for the workers.

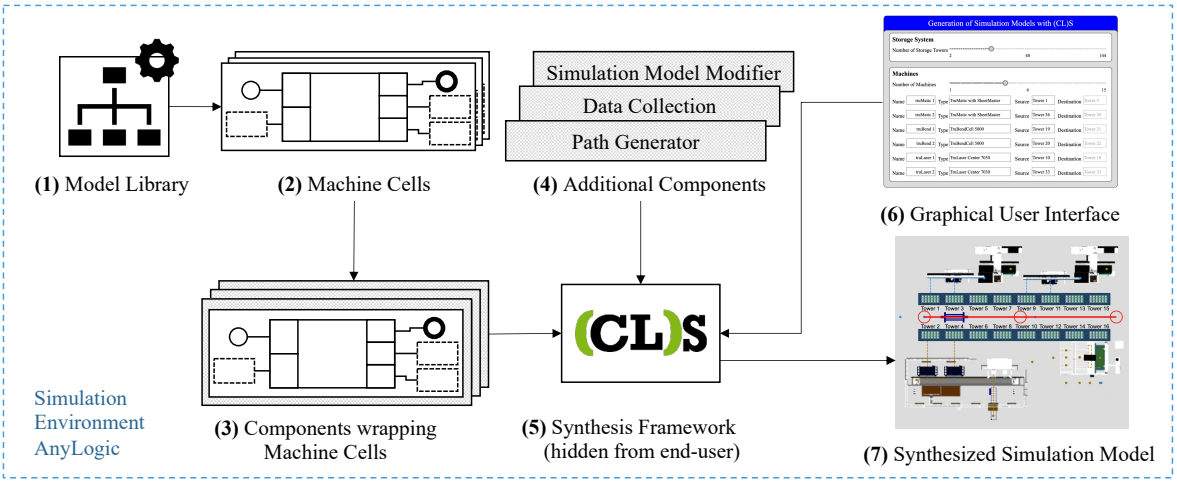

Figure 1: Overview of the architecture and workflow of the proof of concept to synthesize simulation models for shop floors: Machine cells (2), combining machines and automation units from a Model Library (1), are wrapped to form components (3). The Combinatory Logic Synthesizer framework (5) synthesizes simulation models (7) based on this set of components expended with additional components (4) for a given configuration (6). The proof of concept is implemented within the simulation environment AnyLogic.

A graphical user interface (6) is provided to configure the desired simulation model by entering the number of storage towers and the number and types of machines. After configuration, the synthesis can be performed, producing the required simulation model (7). We use the framework Combinatory Logic

Synthesizer (CLS) (5), which implements component-based software synthesis based on combinatory logic with intersection types (Bessai 2019). The synthesis process is not visible to the end-user, but CLS offers various advantages for designing the automated model generation. For instance, CLS supports structural variability in synthesized terms (i.e., programs or models), which permits a flexible specification of model variations. Moreover, this synthesis approach yields a product line instead of a single synthesis result. The semantic layer and the corresponding type specification of components encapsulate domain knowledge to express rules of composition, which is often vital for real-life scenarios. Incorporating components offers a scalable generation method that is extendable and adaptable to further problems by wrapping objects or operations from existing libraries. Given these advantages, CLS is particularly well-suited to implement an automatic simulation model generation. We implemented the PoC within the simulation environment AnyLogic 8 (dotted line) so that a configurable simulation model that contains our implementation can be exported as a stand-alone application without running the simulation tool. Because we can add more simulation buildings block during the migration process, we refer to it as gradual migration.

This PoC is a first step to fulfilling the project's goal with TRUMPF and ITK to establish a web-based tool that enables non-experts to create customer shop floor layouts in a high-end 3D visualization, using the customer's production data as an input to simulate and visualize the production processes. In the future, this should allow a direct analysis of the performance of different shop floor layouts in a short time and without the participation of any expert. Primarily, it allows comparing a large number of variants within a short period to make a sound decision.

This paper is structured as follows: The following section gives an overview of related work on the automatic simulation model generation using component-based software synthesis and other approaches following the configuration idea. Section 3 introduces the theoretical concept of combinatory logic synthesis and presents the CLS framework. Then, Section 4 introduces the industrial use case, which is a shop floor obtaining various machines, their automation units, and a storage system. In Section 5, we present our PoC to synthesize simulation models. Section 6 summarizes the performed experiments using our PoC to illustrate the range of possible solutions. Finally, we discuss the proposed PoC and give an outlook on future work in Section 7.

2 RELATED WORK

There is a considerable amount of literature on automatic simulation model generation. Recent reviews of the literature on this topic, for instance (Reinhardt et al. 2019) and (Wenzel et al. 2019), focus on use cases from production and logistics. Wenzel et al. (2019) address trends and developments in this research field. In this context, the authors emphasize the need for approaches to handle structural variance and propose a solution based on component-based software synthesis.

In subsequent work, Kallat et al. (2020) combine the CLS framework with constraint solving techniques to synthesize simulation models. The simulation models represent factory configurations that contain a fixed number of machines that differ in their settings. The authors use constraint solving for filtering synthesized configurations considering numerical constraints (e.g., total costs or processing time). In (Kallat et al. 2021), the authors use the CLS framework to synthesize simulation models representing block-stacking warehouses. The synthesized simulation models structurally differ in their process logic. In order to enable using the approach without programming knowledge, the authors developed a method for automatically extracting components from existing simulation models. In their approach, the users modify the semantic types of the extracted components and the synthesis goal in a user-friendly web application to achieve new variants. While the previous work synthesized variants of the process logic within an agent, our PoC synthesizes new compositions of agents. Further, we allow using our approach with a graphical user interface without modifying the semantic types of components. Both approaches draw inspiration from the work described in (Heineman et al. 2015), which migrates an existing software library into a product line. This paper also follows the idea but presents a new approach for *gradually* migrating an existing application into a product line.

Some examples of approaches that generate simulation models based on a user configuration are in the literature. Jain and Lechevalier (2016) implemented an approach to generate a multi-resolution simulation model automatically. They are using AnyLogic 8 because it supports various simulation techniques. Our work focuses on agent-based simulation, but we used the CLS framework on the discrete event level in (Kallat et al. 2021). The authors in (Jain and Lechevalier 2016) use different standards, such as ISA-95, ISA-88, or CMSD, depending on the resolution level as input data and configuration. While the idea of a multi-resolution simulation model and using standards as input data are inventive, the authors admit that their approach needs to be enhanced to support more complex simulation models. Our component-based approach scales well to complex models since the complexity is distributed over different components. Moreover, gradually migrating a simulation model enables the progressive generation of more and more parts of a simulation model. Aiming for similar goals to our PoC (Kassen et al. 2021) use AnyLogic to reduce the development time of digital shadows by using a generic model parameterized with production and product data. Wincheringer et al. (2020) developed a generic warehouse simulation model that enables the sales personnel to generate a warehouse variant based on a user configuration. In contrast, we use a master simulation model initially not intended for automatic simulation model generation. Neyrinck et al. (2015) also developed an approach for the sales personnel to generate appropriate simulation model variants. The authors follow a component-based approach, too. They use skills to describe the components' capabilities and the desired simulation model. They developed an algorithm for generating not only the simulation model but also the documentation, wiring diagrams, or customer offers. This seems to be a beneficial approach to support the whole sales process. However, their approach is limited to a specific use case.

3 COMBINATORY LOGIC SYNTHESIS

This section describes the theoretical foundation of the combinatory logic synthesis with intersection types and introduces its practical implementation in the form of the CLS framework.

3.1 Theoretical Concept: Combinatory Logic with Intersection Types

The idea behind combinatory logic synthesis is to synthesize a solution for a given requirement using combinatory logic (Hindley and Seldin 2008) from a set of given typed combinators (representing components) called a repository Γ. Combinatory logic (CL) is based on a language of applicative terms $e ::= X \mid (e\ e)$, where X ranges over combinatory names and $(e\ e)$ represents the application of one term to another (Rehof 2013). To define and control the process of combining components, they have an interface in the form of an intersection type. The intersection type system (Rehof and Urzyczyn 2011) contains type expressions $t ::= A \mid t \to t \mid t \cap t$, where A represents a constant, $t \to t$ a function, and $t \cap t$ an intersection. We use Greek lowercase characters $\sigma, \tau, \ldots$ to symbolize types and Latin capital characters $M, N, \ldots$ for combinatory logic terms. A component X with its related type τ is written $X : \tau$.

For a component, a type can be understood as a property, whereas a function type $A \to B$ defines that a component needs an inputs of type A to work and have property B. An intersection can be seen as an *and* so that a component can have more than one property.

Based on the repository Γ, an inhabitation algorithm tries to find a composition of components that satisfies a given target τ in the form of an intersection type written $\Gamma \vdash ? : \tau$ by using the rules Variable and Modus Ponens defined in equation (1).

$$\frac{X : \tau \in \Gamma}{\Gamma \vdash M : \tau}\ \text{(Variable Rule)} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (M\ N) : \tau}\ \text{(Modus Ponens Rule)} \tag{1}$$

The Variable rule states that if a component X has the type τ in the repository, it can be inhabited from the context that the term $M = X$ has type τ. The Modus Ponens rules state that if a term M has a function type $\sigma \to \tau$ under inhabitation, and it exists a term N with the type σ under inhabitation as an argument, the result of function application $(M\ N)$ has the type τ under inhabitation.

3.2 CLS Framework

The CLS framework (Bessai 2019) implements an inhabitation algorithm for combinatory logic with intersection types in the programming language Scala. It returns an iteration of all found solutions for the specified target and a repository. The components are not restricted to the programming language Scala and can contain any program code. In CLS, it is possible to distinguish between two kinds of types in an intersection. The first one is called a native type and represents the type of a component in Scala (e.g., Int, String, etc.). The second one is called the semantic type and provides the possibility to define additional information and restrictions about a component in the form of a simple string.

The framework was already used in different domains like path planning for machine operations (Schäfer et al. 2021), machine scheduling (Mäckel et al. 2021), modular factory planning (Winkels et al. 2020), and manufacturing simulation models (Kallat et al. 2020).

4 INDUSTRIAL USE CASE

The use case for machine manufacturers like TRUMPF is to demonstrate the performance of their complex sheet metal manufacturing systems. This can be done by comparing different shop floor layouts, running a simulation for each layout in a simulation scenario, and evaluating specific key performance indicators (KPIs) like machine utilization, storage utilization, and throughput. The results are beneficial for dimensioning new factories and optimizing existing factories to support fact-based decision-making (Winkels et al. 2018). The 2D and 3D visualization and animation of the simulation can be convincing for customers investing in new machines or new production lines and results in transparency and a high reliance on the simulation results (Akpan and Brooks 2014).

The shop floors in our scenarios consist of a fully automated high bay storage system with up to 24 storage shelves for each storage tower. Raw material, intermediate or finished products can be stored on pallets on the storage shelves. The number of towers is variable and depends on the shop floor layout. At each tower, one of the following machines from the TRUMPF machine portfolio can be individually placed:

- TruMatic with SheetMaster: An universal machine for punch and laser processes combined with a SheetMaster to automatically load unprocessed material and unload and sort finished parts.
- TruBend Cell 5000: An universal automated bending cell for a wide range of parts.
- TruLaser Center 7030: A fully automated laser cutting machine.

The initial start point of the simulation is a storage system filled with raw material required to fulfill the jobs running on different machines. Therefore the job portfolio can lead to an additional complexity level of the simulation because all KPIs strongly depend on the chosen job portfolio. We aim to find solutions that best fit the customer's job portfolio concerning the chosen combination of KPIs.

5 POC WORKFLOW & SOFTWARE-ARCHITECTURE

Different components work together to synthesize a simulation model based on given configurations and input data. Figure 2 shows the different components grouped in related layers — starting from the layer that contains the Model Library blocks. The latter encapsulates the logic and visualization of all machine models, their automation units, and storage components as re-usable blocks. Based on the Model Library, we introduce an abstraction layer containing predefined machine models representing parametrized machines and their automation units. We define components for the CLS framework based on the abstraction layer. Further, we implemented components that produce paths within the simulation model's visualization. The set of components forms the repository. Based on the repository, CLS synthesizes the runnable simulation model with the target type based on the chosen configuration from the graphical user interface and is executed in the simulation environment. This approach follows the character of internal automatic

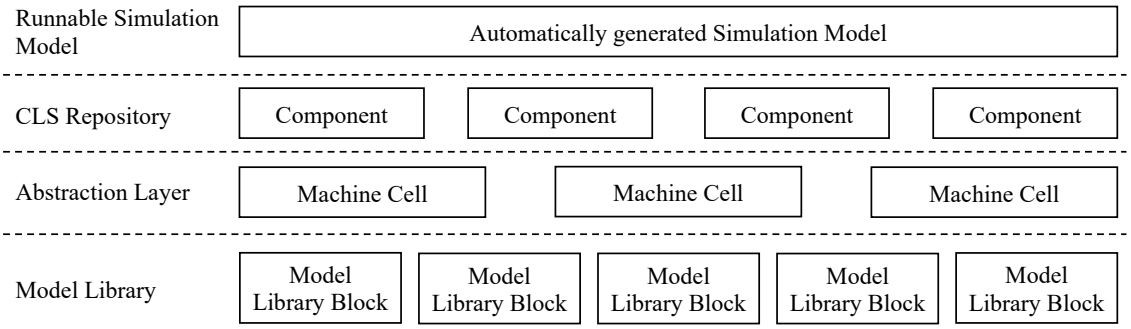

Figure 2: Different elements that work together to synthesize a simulation model and which build on each other - starting from the Model Library building blocks over the machine cells and CLS components to the runnable simulation model.

simulation model generation. This means that the algorithms for generating a simulation model are executed within a simulation environment (Bergmann and Strassburger 2015). The following subsections describe the different parts and their interaction in detail.

5.1 Model Library

Automatic component-based synthesis of user-configured manufacturing simulation models requires a vital standardization of several components to re-use them in different simulation projects. This is done by implementing a Model Library that encapsulates the logic and visualization of all machine models, their automation units, and storage components. Additionally, widely-used logic blocks (e.g., the logic of palletizing parts) are also encapsulated as blocks in the Model Library. Simulation experts can use these blocks and build their simulation models without creating machine models on their own. Model Library blocks can be parametrized to be used in different scenarios and cover all common variations of the TRUMPF portfolio.

5.2 Abstraction Layer

As mentioned before, the Model Library is built up in a very modular way. The initial scope of this use case considers predefined arrangements of Model Library elements to incorporate an abstraction layer that exposes reduced complexity. This layer contains machine cells that represent specific machine configurations. For instance, one machine cell wraps a TRUMPF cutting machine with a TRUMPF automation unit for automatically loading and unloading the machine and a storage cart to deliver or return material from and to the storage. We implemented these machine cells manually in the simulation tool using the corresponding Model Library blocks. Then, we wrap the machine cells into components that are used for component-based synthesis. This is a one-time effort.

5.3 Repository and Target Type

The repository consists of 14 software components covering different parts of the simulation model. A reference to the simulation model's root agent allows software components to access model variables. The components' implementation uses this reference to read model-specific data and adapt their embedded code. Moreover, we can add new elements to the model by integrating a machine cell factory (in the sense of object-oriented programming) in the simulation model to create new agents. The software components can access the information of the user-defined configuration and use it to adjust their implementation.

The **top-level component** generates and integrates different parts of the model. It manipulates the model directly and additionally exposes selected model contents. Due to a missing API functionality, some

operations can only be performed in the scope of the simulation tool, i.e., they are not admissible through external code. Therefore, the synthesis procedure uses a context data object that holds relevant model information. It contains machine cells and adapted simulation objects required for the current model.

$$\begin{aligned} &\mathit{Context} \cap \mathit{IsComplete} \to \mathit{TruMaticMachineCell} \to \mathit{TruBendCellMachineCell} \to \\ &\mathit{TruLaserCenterMachineCell} \to \mathit{InputData} \to \mathit{TopLevelComponent} \end{aligned} \tag{2}$$

The semantic type of this software component is shown in equation (2). This type expression states that the term yields a program with type `TopLevelComponent` when supplied with suitable arguments. These consist of a simulation model context object, three lists of machine agent objects, and the user configuration.

The **context** is a central aspect of collecting the changes required to handle the side-effects of software components. The data structure contains a use-case-specific selection of the information required to manipulate the simulation model. For instance, it provides an object of the class `ConfigSetup`, which holds the relevant information to initialize the given simulation models. The class is part of the model library, and the repository can integrate this library due to Scala's JVM compatibility. Furthermore, the context integrates worker agents and a corresponding graphical element for visual representation, which is used to define material flows, resource flows, and the construction of the visual paths. The context also references the UI modeling areas as the simulation engine requires this information when adding new elements to the model. Moreover, it references a storage object of the model library and contains references to the included machines as a list of AnyLogic agents.

$$\begin{aligned} &\mathit{Context} \cap \mathit{IsEmpty} \to \mathit{BasicStorage} \cap \mathit{HasStorageTowers} \to \mathit{ConfigSetup} \to \\ &\mathit{WorkerHome} \to \mathit{Worker} \to \mathit{ViewArea} \to \mathit{Context} \cap \mathit{IsComplete} \end{aligned} \tag{3}$$

The top-level component in equation (2) requires a context object that holds the required data for the model construction. The semantic types *IsEmpty* and *IsComplete* distinguish between context states, where the semantic type *IsComplete* denotes an object that results from an initialization procedure. A software component encapsulates this procedure, and its semantic type is displayed in equation (3). It requires an empty context object and the context data as arguments, and it returns a complete context object that contains suitable values provided by the repository.

Every **machine type** has its software component that fills a list with agents according to the user configuration. It requires the user configuration as an argument and uses the contained information to construct the list. The list of wrappers is used to call the simulation model's machine cell factory, which adds agents to the corresponding agent population in the process. After generating these agents, the software components manipulate their properties to set the agents' position and orientation.

5.4 Integration in Simulation Environment

The superior goal is to provide an application that allows executing the simulation model without installing the simulation tool or other programs. Therefore, we can use the AnyLogic 8 export functionality to export a simulation model into a standalone application. We can include the synthesis framework and the model library as dependencies. In the following subsection, we introduce the simulation model that is exported. Then, we present the graphical user interface that allows a user to configure the factory shop floor, perform the synthesis, and run the simulation.

5.4.1 Template Simulation Model

We implemented a simulation model that we use as a template. To use the Model Library and the synthesis module, we add both as dependencies to the simulation model. The simulation model template contains *static* elements and *dynamic* elements. Static elements occur in every variant of the simulation model

and shall not be modified. An example for static elements are resource pools within a simulation model. In our use case, we have a resource pool that contains workers. The number of workers is identical in each configuration. Nevertheless, we need to reinitialize static elements after synthesis because the model structure has changed. The dynamic elements, for instance, machine cells, differ in each variant depending on the user configuration. We synthesize the dynamic elements and add them during the synthesis runtime or simulation runtime.

However, we recognize that adding individual machine cells after the simulation has started is hardly realizable in the used simulation tool. As a solution, we use populations that store the dynamic elements. Populations are comparable to lists, which can solely be filled with a pre-defined agent type. Therefore, our simulation model template contains several populations for each machine type. All populations are initially empty. After or during synthesis, we add the synthesized machine cells into the corresponding population.

5.4.2 Graphical User Interface and Input Data

To reduce the costs per simulation model, people with little knowledge about simulation should be able to create custom shop floor layouts, run the simulation and visualize the production process. This requires a simple to use graphical user interface (see Figure 3) that makes it possible to generate various simulation models depending on the number of selected storage towers and different types of machines. In the scenario we consider in this PoC, up to 144 storage towers can be placed in two rows connected by a storage-retrieval machine. The PoC supports the three types of machines from the machine portfolio of TRUMPF TruMatic, TruBend Cell 5000, and TruLaser Center 7030. Each machine has an individual name for identification and a source and destination storage tower to collect raw material and store the processed parts. A user can specify only the source storage tower. The destination storage tower is automatically set depending on the type of the machine and so the resulting space it needs. In this concept, the punch laser machine and bending machine use the storage tower next to the source storage tower, whereas the laser machine uses the same store towers for source and destination. To run the simulation model, it needs additional information about the different jobs the machines should execute. In this case, an XML file is defined with definitions of different jobs and process steps the different machines should perform.

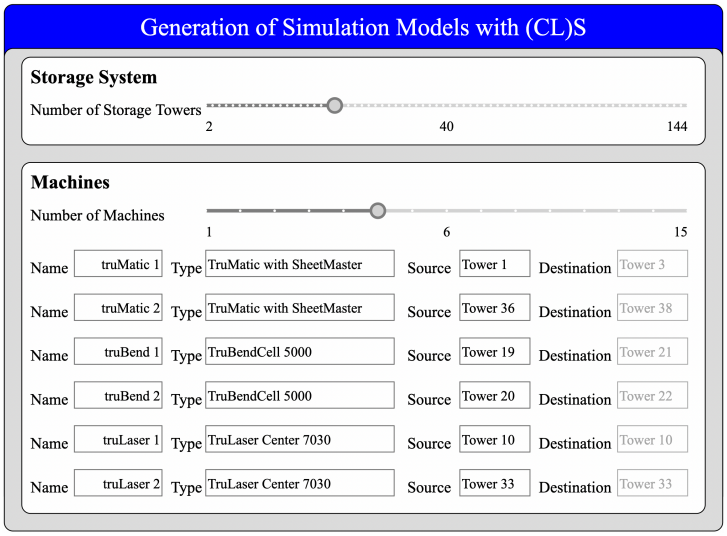

Figure 3: Graphical User Interface to control and specify the synthesized simulation model. The displayed case shows the configuration for a simulation model with 40 storage towers, two TruMatic with SheetMaster, two TruBend Cell 5000, and two TruLaser Center 7030 of the company TRUMPF.

5.5 Generating Simulation Models

The process of generating a simulation model variant takes place in the simulation model as well as in the CLS module. Figure 4 summarizes the process of generating a simulation model. First, a user configures the desired system within the graphical user interface of our simulation model. After starting the simulation, the startup routine of the simulation model instantiates our CLS implementation. Then, the startup routine calls our synthesis routine and passes the user configuration and a reference on the root agent as arguments. The synthesis routine instantiates the repository and performs the synthesis using the top-level component's target goal as synthesis goal. As mentioned before, the synthesis generates machine cells and performs adaptions on the simulation model by adding them to the corresponding populations. As already said, we call several initialization functions after synthesis. Then, the simulation starts and performs the simulation. We want to emphasize that we do not require any further user input after the configuration.

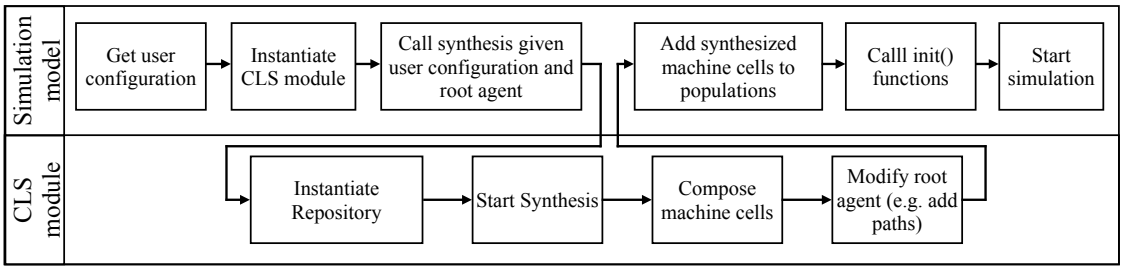

Figure 4: Process of automatic simulation model generation with Combinatory Logic Synthesizer.

6 EXPERIMENTS

We perform experiments using our PoC to demonstrate the ability to generate various configurations that differ in their number of machines and storage towers. The number of each machine type varies from 2 to 5 resulting in $4 \times 4 \times 4 = 64$ configurations. Further, we vary the number of storage towers by the factors 1.0 and 1.2, which results in $64 \times 2 = 128$ total configurations. Figure 5 shows four exemplary 2D visualizations of shop floor simulation models. We use the same set of jobs for all shop floor layouts and simulation scenarios to reduce the complexity. It contains 30 jobs irrespective of the number of machines. This leads to a situation where the machine utilization is higher in a configuration with fewer machines because each machine processes more jobs. Since the simulations show no stochastic effects, we achieve deterministic simulation results. Therefore, it is sufficient to run a simulation once for each configuration.

The workflow is identical as described in Section 5; the only difference is that we automate the user-configuration process using the experiment functionality in AnyLogic 8. We enhanced our template simulation model with a function that automatically assigns machines to storage towers considering the required space of each machine and ensuring a valid layout. Further, we determine the initial number of storage towers, raw material, and pallets considering the number of machines in a configuration. This allows us to generate user configurations as input for our synthesis and to test them by running the corresponding synthesized simulation model. Each simulation run stops when all jobs are finished.

Table 1 shows the simulation results of selected configurations. Each row represents a configuration with the number of machines and storage towers, the utilization of each machine type, and the utilization of the storage-retrieval machine.

In our work, we implemented the PoC with an emphasis on demonstrating the ability to start with an existing master simulation model to generate a large number of variants. Therefore, a detailed analysis of the simulation results and a subsequent optimization concerning selected target functions are left for future work. Nevertheless, the results lead to initial conclusions for our fictive job portfolio. For instance, the results indicate that the single storage-retrieval machine seems to be the bottleneck since the utilization

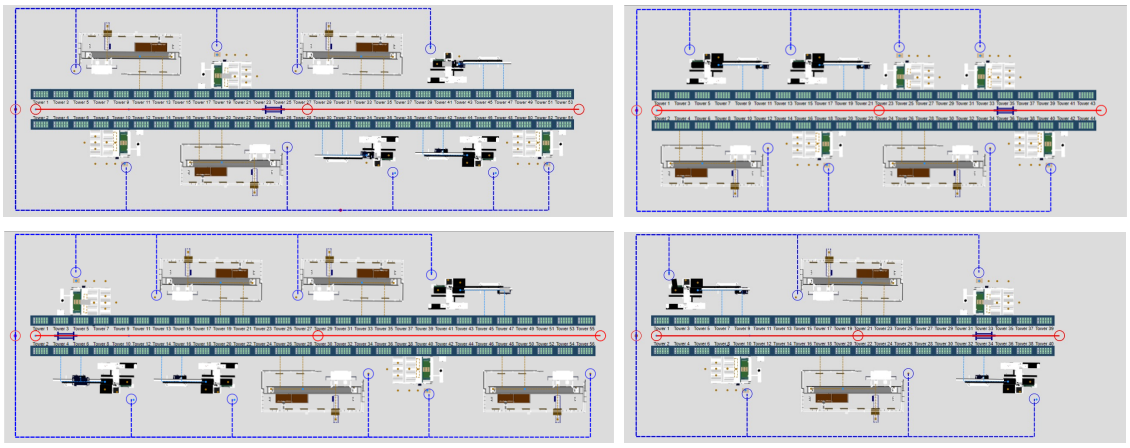

Figure 5: Screenshots of the 2D visualization of generated simulation model variants which represent shop floors with various instances of each machine type and a storage system with a varying number of towers. The paths for the workers are also synthesized.

Table 1: Results of the simulation runs of synthesized simulation models. The results contain the number of machines and storage towers, the utilization of the different machines, and the utilization of the storage-retrieval machine (SRM) for each configuration.

Shop Floor Configuration				Machine Utilization			
TruMatic	TruBend Cell	TruLaser Center	Storage Towers	TruMatic	TruBend Cell	TruLaser Center	SRM Utilization
2	2	2	48	52.8 %	80.0 %	71.2 %	72.2 %
2	2	2	58	49.9 %	83.7 %	70.0 %	73.2 %
3	4	2	72	47.3 %	69.8 %	66.0 %	78.7 %
3	4	2	86	51.2 %	69.5 %	62.9 %	80.3 %
5	5	5	120	30.1 %	61.5 %	45.7 %	91.6 %
5	5	5	144	30.9 %	62.8 %	51.9 %	93.0 %

is very high while the utilization of the machines is very low and diminishing in scenarios with a higher number of machines. This leads to situations where machines have to wait for material, which decreases their utilization.

The whole generation process (including synthesis) of a simulation model variant took three seconds on average, and the complete process of generating and simulating 128 configurations took seven minutes. We ran the experiment on a workstation computer equipped with an Intel i7 processor and 64 GB RAM.

7 CONCLUSION AND FUTURE WORK

This paper presents a proof of concept for gradually migrating a master simulation model for shop floor layouts of machines into a product line of different simulation models to explore and find suitable solutions. The approach uses combinatory logic with intersection types to define a set of components that forms the basis for the Combinatory Logic Synthesizer framework to synthesize new solutions. To control and specify the synthesis process's configuration, we provide a graphical user interface for easy use without any knowledge about simulation design or combinatory logic. Further, we performed several simulation experiments to demonstrate various possible solutions for a given specification based on the logical dependencies and structural variance. The downside of this approach is the needed additional up-front work and time to wrap code in components and specify the semantic layer of the repository. However, these costs are amortizable

for problem domains that expose a high degree of variability. In addition, these abstractions make adding additional components like new machines, storage systems, or other needed agents easy without interfering with existing ones.

Based on the experience of this work and to drive automation even further, we plan to automate phases of a simulation study, such as data collection or analysis of simulation runs. Moreover, we investigate to automate the wrapping of the simulation building blocks components in the sense of (Kallat et al. 2021). For instance, our proof of concept could overtake a part of this wrapping by automatically composing Model Library blocks into machine cells. This would compensate for the up-front costs for creating, specifying, and wrapping the components. Combinatory Logic Synthesizer can be used to enable optimization procedures using constraint solver (Kallat et al. 2020) and machine-learning techniques. Schäfer et al. (2022) propose a methodology to explore the algorithmic design space of sampling-based motion planning programs. Similarly, future work could aim to synthesize and execute simulation models in an active-learning loop. The extracted KPIs train a machine-learning model to determine optimal manufacturing configurations and solutions.

ACKNOWLEDGMENTS

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

REFERENCES

Akpan, I. J., and R. J. Brooks. 2014, August. "Experimental Evaluation of User Performance on Two-dimensional and Three-dimensional Perspective Displays in Discrete-event Simulation". *Decision Support Systems* 64:14–30.

Bergmann, S., and S. Strassburger. 2015. "On the use of the Core Manufacturing Simulation Data (CMSD) standard : experiences and recommendations". 12.

Bessai, J. 2019. *A Type-theoretic Framework for Software Component Synthesis*. Ph. D. thesis, Technical University of Dortmund, Germany.

Chance, F., J. Robinson, and J. W. Fowler. 1996, November. "Supporting Manufacturing with Simulation: Model Design, Development, and Deployment". In *Proceedings of the 28th conference on Winter simulation*, WSC '96, 114–121. USA: IEEE Computer Society.

Heineman, G., A. Hoxha, B. Düdder, and J. Rehof. 2015. "Towards Migrating Object-Oriented Frameworks to Enable Synthesis of Product Line Members". In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, 56–60. New York, NY, USA: Association for Computing Machinery.

Hindley, J. R., and J. P. Seldin. 2008. *Lambda-calculus and Combinators, an Introduction*. Cambridge University Press.

Jain, S., and D. Lechevalier. 2016. "Standards Based Generation of a Virtual Factory Model". In *Proceedings of the 2016 Winter Simulation Conference*, WSC '16, 2762–2773: IEEE Press.

Kallat, F., C. Mieth, J. Rehof, and A. Meyer. 2020, January. "Using Component-based Software Synthesis and Constraint Solving to generate Sets of Manufacturing Simulation Models". *Procedia CIRP* 93:556–561.

Kallat, F., J. Pfrommer, J. Bessai, J. Rehof, and A. Meyer. 2021, January. "Automatic Building of a Repository for Component-based Synthesis of Warehouse Simulation Models". *Procedia CIRP* 104:1440–1445.

Kassen, S., H. Tammen, M. Zarte, and A. Pechmann. 2021. "Concept and Case Study for a Generic Simulation as a Digital Shadow to Be Used for Production Optimisation". *Processes* 9(8).

Lugaresi, G. 2021, 10. *Automated Generation and Exploitation of Discrete Event Simulation Models for Decision Making in Manufacturing*. Ph. D. thesis, Politecnico di Milano, Italy.

Mäckel, D., J. Winkels, and C. Schumacher. 2021. "Synthesis of Scheduling Heuristics by Composition and Recombination". In *Optimization and Learning*, edited by B. Dorronsoro, L. Amodeo, M. Pavone, and P. Ruiz, Communications in Computer and Information Science, 283–293. Cham: Springer International Publishing.

Negahban, A., and J. S. Smith. 2014, April. "Simulation for Manufacturing System Design and Operation: Literature Review and Analysis". *Journal of Manufacturing Systems* 33(2):241–261.

Neyrinck, A., A. Lechler, and A. Verl. 2015, January. "Automatic Variant Configuration and Generation of Simulation Models for Comparison of Plant and Machinery Variants". *Procedia CIRP* 29:62–67.

Rehof, J. 2013. "Towards Combinatory Logic Synthesis". In *1st International Workshop on Behavioural Types, BEAT*.

Rehof, J., and P. Urzyczyn. 2011. "Finite Combinatory Logic with Intersection Types". In *TLCA*, Volume 6690 of *Lecture Notes in Computer Science*, 169–183: Springer.

Reinhardt, H., M. Weber, and M. Putz. 2019. "A Survey on Automatic Model Generation for Material Flow Simulation in Discrete Manufacturing". *Procedia CIRP* 81:121–126. 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019.

Schäfer, T., J. Bessai, C. Chaumet, J. Rehof, and C. Riest. 2022. "Design Space Exploration for Sampling-Based Motion Planning Programs with Combinatory Logic Synthesis". In *Algorithmic Foundations of Robotics XV*. Cham: Springer International Publishing.

Schäfer, T., J. A. Bergmann, R. G. Carballo, J. Rehof, and P. Wiederkehr. 2021. "A Synthesis-based Tool Path Planning Approach for Machining Operations". *Procedia CIRP* 104:918–923. 54th CIRP CMS 2021 - Towards Digitalized Manufacturing 4.0.

Wenzel, S., J. Stolipin, J. Rehof, and J. Winkels. 2019. "Trends in Automatic Composition of Structures for Simulation Models in Production and Logistics". In *Proceedings of the Winter Simulation Conference*, WSC '19, 2190–2200: IEEE Press.

Wincheringer, W., M. Sekulic, and M. Kexel. 2020. "Generisches Simulationsmodell für automatische Hochregallagersysteme". In *Proceedings ASIM SST 2020*, 389–395: ARGESIM Publisher Vienna.

Winkels, J., J. Graefenstein, L. Lenz, K. C. Weist, K. Krebil, and M. Gralla. 2020, January. "A Hybrid Approach of Modular Planning – Synchronizing Factory and Building Planning by Using Component based Synthesis". In *Proceedings of the HICSS 2020 : Hawaii International Conference on System Sciences*. Hawaii, USA.

Winkels, J., J. Graefenstein, T. Schäfer, D. Scholz, J. Rehof, and M. Henke. 2018. "Automatic Composition of Rough Solution Possibilities in the Target Planning of Factory Planning Projects by Means of Combinatory Logic". In *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*, edited by T. Margaria and B. Steffen, Lecture Notes in Computer Science, 487–503. Cham: Springer International Publishing.

AUTHOR BIOGRAPHIES

ALEXANDER MAGES is a simulation expert at TRUMPF Werkzeugmaschinen SE + Co. KG. in Ditzingen, Germany. He holds a M.Sc. degree in Autonomous Systems from the University of Stuttgart. He works as simulation engineer for manufacturing simulation in the sheet metal industry. His email address is alexander.mages@trumpf.com.

CARINA MIETH is a Ph.D. student in mechanical engineering at the TU Dortmund University. She holds a M.Sc. in electrical engineering from the Karlsruhe Institute of Technology. She is an associated member in the DFG-funded research training group adaption intelligence of factories in a dynamic and complex environment. Her research focuses on cyber-physical production systems and simulation input modeling for manufacturing simulation. Her email address is carina.mieth@trumpf.com and her website is https://www.linkedin.com/in/carina-mieth/.

JENS HETZLER is simulation expert at ITK Engineering GmbH, a 100% subsidiary of Robert Bosch GmbH in Rülzheim, Germany. He holds a Ph.D. degree in Physical Chemistry from the Karlsruhe Institute of Technology (KIT) and works for more than 5 years as a software engineer and project leader for several customer projects. His main fields are solving Industry 4.0 related customer issues with the help of factory simulations. His email address is jens.hetzler@itk-engineering.de. His website is https://www.itk-engineering.de.

JAKOB REHOF is professor of Computer Science, Chair of Software Engineering, at the TU Dortmund University, and director of research strategy at the Fraunhofer Institute for Software and Systems Engineering (ISST) Dortmund. Jakob Rehof got his Ph.D. in Computer Science at the University of Copenhagen. His main field of research is the automatic synthesis of programs from component collections using combinatory logic. His email address is jakob.rehof@tu-dortmund.de.

FADIL KALLAT is a research assistant and Ph.D. student at the Department of Computer Science at the TU Dortmund in Dortmund, Germany. He works at the Chair for Software Engineering with his research focus on the synthesis of simulation models. His email address is fadil.kallat@tu-dortmund.de and his website is https://www.linkedin.com/in/fadilkallat/.

CHRISTIAN RIEST is a research assistant and Ph.D. student at the Department of Computer Science at the TU Dortmund in Dortmund, Germany. He works at the Chair for Software Engineering with his research focus on the connection between program synthesis and machine learning ideas. His email address is christian.riest@tu-dortmund.de.

TRISTAN SCHÄFER is a research assistant and Ph.D. student at the Department of Computer Science at the TU Dortmund in Dortmund, Germany. He works at the Chair for Software Engineering with his research focus on the synthesis of motion planning algorithms. His email address is tristan.schaefer@tu-dortmund.de.