Simulation of Push- and Pull-Processes in Logistics

Usage, Limitations, and Result Presentation of Clock Pulse and Event Triggered Models

Carlo Simon, Stefan Haag, and Lara Zakfeld

Hochschule Worms
Erenburgerstr. 19, 67549 Worms, Germany
Email: {simon,haag,zakfeld}@hs-worms.de

***Abstract*—The change from a push to a pull strategy constitutes a considerable intervention in the operational logistics and has effects on procurement, the design of processes and the evaluated, internal inventories. Simulation models that anticipate the consequences of such a change need to fulfill two main objectives: 1.) They have to represent the system behavior over time, but in time lapse instead of real time. To achieve this, two approaches can be used, the first of which is a clock pulse simulation where the system's state is calculated for every discrete time step, e. g. each second. The second one is an event triggered simulation where only those points in time are computed at which an actual change occurs. Both methods have benefits and drawbacks as is elaborated. 2.) They need to take into account actual production data and, in order to use this data, implement decision rules. These aspects can be realized with Petri nets that the authors use as preferred modeling language for decades, because Petri net models are illustrative and can be executed or simulated, respectively. A novel, web-based Petri net modeling and simulation environment - the Process-Simulation.Center - allows for training modelers and testing different procedures and techniques of model generation. Using a teaching laboratory for logistics as a sample application, clock pulse and event triggered simulation models are demonstrated, as well as how they can be developed, how they have to be interpreted, and which possible obstacles have to be considered. Concretely, the consequences of switching logistics processes from push to pull principles are regarded concerning the storage costs. This paper demonstrates the interplay between new modeling approaches with the aid of Petri nets and the novel tool without which these models would not have been possible.**

***Keywords*—Conceptual modeling of timed dynamic systems; Clock Pulse Simulation; Event Triggered Simulation; Petri nets; Logistics.**

I. Introduction

This paper is a revised and extended version of a contribution to *SIMUL 2020: The Twelfth International Conference on Advances in System Simulation* [1].

Change is an integral part in every organizational context. One option to examine varying approaches and their differing results is simulation. Depending on the circumstances there may exist various foci. Hence, simulation models and their implementation differ in accordance to these goals.

For example, reducing costs while at the same time increasing the production's flexibility is a combined goal for manufacturers. Beside an investment in better and faster machines, rethinking production strategies and processes is also feasible.

Changing production from push to pull is one (possibly cheap) option. The advantages have been demonstrated in many production lines. Nonetheless, push strategies are still widely in practical use. What is the reason for this? The authors assume that producers are uncertain about the consequences of such changes. In this case, conceptual and simulatable models of the current and intended production lines could objectify decisions on the reorganization of production. Depending on the objective, different information, models and simulations are needed. How to choose between two modeling methods is subject of Section VIII where usage and some limitations of these methods are also discussed.

Petri nets are used to present two approaches for implementing corresponding models. As examined in Section VII, one is event triggered simulation that eventually leads to results where the final outcome is more important than the path to this outcome [1]. Demonstrated in Section VI is the other one, namely clock pulse simulation whose results allow for closer examination of the actual process execution [2]. Section V outlines basic considerations for IT-based simulations.

The setting for these models is a training laboratory for logistics students at the Worms University of Applied Sciences. The so-called *Box Game*, which is introduced in Section IV, was developed to impart knowledge and practical experience in highly relevant logistics processes while at the same time maintaining a relatively simple structure. It is ideally suited to explore different possibilities of conceptual modeling and simulation. In Section III the methodological approach used by the authors in the context of modeling and simulation with the Process-Simulation.Center (*P-S.C*) is described. A part of the directly following Section II about related work considers the reasoning as to why the tool's development cannot be separated from the modeling and simulation approach discussed.

II. Related Work

Since this paper combines conceptual modeling with Petri nets and the simulation of laboratory processes in logistics, related work for both fields is considered.

A. Push, Pull and Kanban

In a push production, every workstation produces as soon as being supplied sufficiently regardless of a given demand. This leads to a steady production and a high utilization rate.

In a pull production, the workstations only produce for a given actual demand, resulting in lower stocks and a more flexible production. Which of these paradigms is advantageous over the other depends on the circumstances. Sometimes, mixed solutions are best [3].

Kanban is a method to realize pull principles in logistics and, hence, to lower unnecessary stocks by controlling the replenishment of material to be processed. If a threshold is recognized, a kanban signal initiates a pull request that includes information on the batch size, leading to stable production sizes. Dependening on the used variant, the replenishment can be controlled by cards, empty containers, via e-kanban or by use of a supermarket system [4], [5].

As the pull requests establish an order chain starting at the dispatch warehouse, information in kanban systems flows upstream, while material flows downstream. Different types of kanban may be used to account for the type of material, set up times for production or relevant internal factors [6].

B. Petri Nets

Petri nets can be used to study the performance of push and pull approaches. Practically highly relevant constraints like manufacturing and setup times, vehicle routing or concurrent processing become operational and, thus, flexible manufacturing systems can be examined [3]. Large and interlocked systems can be modeled by expanding on local components; applying different Petri net specifications suited for respective tasks is beneficiary [7].

Originally, Petri nets are defined as Place/Transition nets (P/T) with anonymous tokens indicating a system's state [8]. Diverse concepts for representing high level information in Petri nets exist, the most widely known being the following:

Predicate/Transition nets (Pr/T) omit anonymous tokens for ones carrying data that can be processed and altered by use of functions encoded on transitions. When firing, these functions accept data from tokens on the preset. Functions return their results by putting appropriate tokens on the postset. The places serve as predicates according to which transitions may fire. Thus, it is possible to model interactions of tokens according to real-world influences or with each other [9].

Colored Petri Nets (CPN) integrate colors into Petri nets such that tokens, places and transitions have an assigned identity, their color. When determining if a transition is enabled, the adjacent places and their tokens are examined by color separately. This allows for more compact net representations under certain circumstances [10], [11].

C. Time Concepts in Petri Nets

Since time is an important dimension for the modeling of processes and dynamic systems in general, there exist numerous approaches for handling time aspects in Petri nets. They differ concerning their expressiveness (discrete or continuous time) and which Petri net elements are used to express time constraints (places, transitions, arcs, or tokens).

One possible implementation puts one or two time values on transitions, the lower being a delay up to which the transition is not enabled while the higher presents the latest possible moment of firing. This may lead either to a forced firing, the reset of a clock (where the lower value describes a kind of preparation time and the higher one an expiration time from when a new preparation needs to be conducted) or even to a dead net. Variations include time consuming firing [12] as well as firing without time consumption [13].

Another obvious possibility is to assign time values to the places, again representing lower and upper bounds. These bounds represent the availability of tokens, either as delay until a token becomes available [14] or as time windows in which they are available [15], [16].

Yet another possible implementation is to define the permeability of arcs relative to the moment an adjacent place was marked or an adjacent transition was enabled. The cited concepts are equivalent [17]. However, they have the disadvantage that the state of such nets does not only rely on the respective markings, but also on some kind of timer clocks.

D. Timestamps and Petri Nets

Timestamps are a means to encode time data in the marking.

Timestamp Nets introduce tokens with (only) timestamps as information designating the moment the corresponding token was placed. Transitions may fire in time windows as given by two non-negative values on the transitions' incoming arcs [18]. The permeability of arcs depends on these timestamps [19].

Extended Timestamp Nets integrate the concepts of Pr/T and Timestamp nets such that tokens carry timestamps and any further information [20].

In (extended) timestamp nets, a separate clock becomes unnecessary. Durations or important points in time can be processed by means of simple algebraic operations.

A potential drawback of all described time-dependent Petri net concepts could be a more complex situation of enablement where a transition bearing a token can be enabled, but also (depending on the arcs' permeability) not yet or no longer. This may possibly lead to a user's perception of a quasi-existence of different markings at the same time.

Some of the approaches presented in Sections II-C and II-D may be transformed into each other quite effortlessly [21], [22]. All of the presented Petri net formalisms, however, use artificial, abstract time units. To model and simulate real-world applications, real time values should be used. To this end, actual date and time data types seem beneficial to be included as possible information on tokens. Such nets can be regarded as natural extension of Pr/T nets, allowing for both time calculations inside the model and controlling the execution.

E. Further Modeling Approaches

For reference, there are other modeling methods that were developed to combine time and process structures.

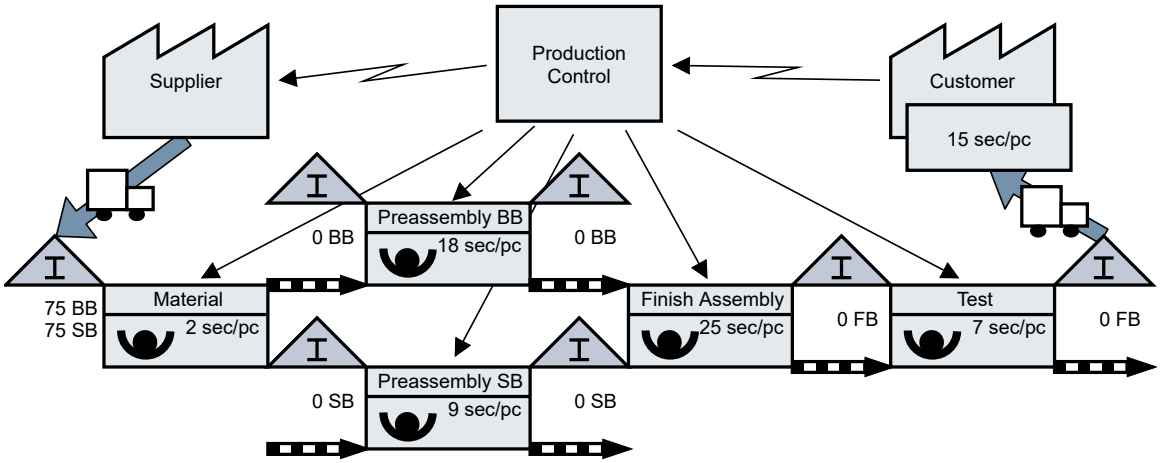

Fig. 1. Value stream diagram of the *Box Game*.

Value Stream Diagrams (VSD) establish models of flows of information and material in order to evaluate value streams. To optimize the value streams, wait times (beside other factors) need to be minimized. The value stream method exposes such wait times. This concept became widely known due to Toyotas Production System from 1930 and its advancements by Japanese engineers Taiichi Ohno and Eiji Toyoda, but dates back as far as 1914 when graphical nets were used to examine routings and other flows to help *Installing Efficiency Methods* in a manufacturing company [23], [24].

Figure 1 shows the scenario explained in Section IV as a VSD. It gives a suitable overview of the value stream from customer to supplier, yet cannot be simulated.

Business Process Model and Notation (BPMN) is a notation and representation language for modeling business processes. It is extensively used due to the relative ease of both creating and understanding models. Using BPMN, it is possible to create both high-level models of companies and low-level models of single processes in a graphical approach similar to flowcharts [25], [26].

Although there are similarities between BPMN and Petri nets, the former lack the mathematical toolset that can be used to analyze Petri nets in form of linear algebra.

F. Process-Simulation.Center

To develop conceptual models for process simulation or execution, tools are needed beside the formal mathematical base. The *P-S.C* supports development of P/T and Pr/T nets [27]. It is possible to assign data types to places and use these in analogy to database tables. Own types for date and time are substructures for the simulation of processes in production and logistics and enhance the approaches to timed Petri nets.

In contrast to relational algebra and SQL where operations like select or project are applied to the set of all affected tuples and result in a set again, in *P-S.C* the tuples are processed serially. This is since in business and production, work items are also treated one after another.

Supporting high-level Petri nets, the P-S.C facilitates the definition of individual data objects as tokens such that data-driven process simulations can be conducted within the tool. A decision on the concrete sequence is made locally by the transitions of the net which also have the ability to aggregate over tuple tokens on a place, a functionality know from database systems.

Also, the *P-S.C* can be used to combine the process view on a system with other views. Process maps can be used to collate different processes with each other and to express the strategic value of processes as primary, supportive or managing. The organizational structure of an institution can be combined with the Petri net view on the processes by assigning its nodes to swim lanes for the responsible organizational units. Organizational charts complete the functions of the *P-S.C*.

Contrary to most other conceptual modeling tools, especially those that have been designed for Petri nets, a specification language has been developed for the *P-S.C* with which all types of models are scripted. Due to strong algorithms for automatic layout, modelers can concentrate purely on structural aspects of the domain to be expressed.

The dearth of current Petri net tools, the quaint user experience of most of the still working ones and the unique approach of using textual programming instead of drag-and-drop modeling in combination with the added functionality are the main reasons for the implementation of the *P-S.C*.

The models presented here could not have been implemented without the *P-S.C* as no tool with comparable capabilities is known to the authors. Also, the models cannot be properly examined without (at least some) insights into the tool. Thus, it is not sensible to separate the models from the tool as is explained further in Sections V, VI and VII.

III. Methodology

Researching new and comparing different modeling techniques for (high-level) Petri nets with individual tokens relies on the existence of proper modeling and simulation tools.

The *P-S.C* is in development for several years following the guidelines for design science research according to [28] whose main point is to create a feasible artifact. The following is a brief outline of the thereby conducted research:

Design as an Artifact: The *P-S.C* is a web-based specification and simulation software for processes encompassing both user defined and primitive data types, organizational structures and process maps. Process execution can be controlled by business relevant data linked via an interface. Also, sensors and actuators of a Raspberry Pi can be used in case the tool is installed on such a device.
The established models are a second artifact, as they are used in a teaching environment at university level but also for vocational training. They aid in transferring knowledge of modeling and simulation techniques.

Problem Relevance: Examining possible consequences of change is a highly relevant task for every company. The *P-S.C* provides practitioners with the means to model, simulate, and optimize processes with high-level Petri nets in a powerful and contemporary user experience.
A simulation permits the extension of real-world experiences in a learning environment. It helps to overcome typical limitations concerning time, resources, space, and people. The simulation environment, however, must be generic enough to assure the intended learning success. From a conceptual modeling perspective, it must be determined if all these aspects can be expressed and simulated due to a formal, semantic base.

Design Evaluation: The *P-S.C* has already been used by companies in logistics and trade. Students of an integrated logistics degree program developed a simulation model for the reorganization of a returns process [29]. The tool is also used for problem-based and research-oriented learning in bachelor and master degree programs [30].

Research Contribution: The *P-S.C* is a practical application on the theoretical basis of high-level Petri nets combined with views on organizational structures, process maps and data types. The tool offers a novel user experience and provides new insights in Petri net based modeling. As an abstract concept Petri nets do not force specific modeling approaches such as flow diagrams, value stream diagrams or other pictorial modeling approaches.

Research Rigor: The benefits of a simulation approach in opposite to pure visual methods is evaluated in mentioned bachelor and master courses as well as in cooperation with partner companies of integrated degree programs.

Design as a Search Process: Both presented prototype and models are the latest in a series that starts from the initial implementation of the underlying principles and ends in a productive system. Each implementation step has been evaluated and published (for instance, [31], [32], [33]).

Communication of Research: The results achieved so far are relevant for both research and practice. They are presented on pertinent conferences but also, more eidetic, for students and practitioners in advanced training programs.

IV. A Simulation Laboratory for Processes in Logistics

The so-called *Box Game* has been developed by Prof. Dr. Christian Reuter at the Worms University of Applied Sciences to teach students in logistics and is used as a sample application (cf. [1], [2]). Despite its simplicity, very different kinds of processes that also have a high impact for practice can be observed. It is, therefore, ideal for trying out different ways of conceptual modeling and simulation.

The concrete example is a simple construction process where students assemble small and large boxes, put the smaller into the larger ones, and finally check the quality. Thus, characteristics of push and pull systems are illustrated.

Training members are present during the game. Ideally, they take part in the game in order to gain first hand experience of motivation and work situations. Being engaged in the training helps the students to recognize different types of waste (so-called muda according to [24]), such as *overproduction*, *waiting*, and *motion*, but also the transformation of waste types. Finally, discussing the shared experiences is a major part of the learning success. A complete simulation run of the *Box Game* lasts approximately two to three hours.

Despite the simplicity of the used material and the low level of technical requirements, the *Box Game* is easily transferable to assembly work stations in a more generalized form and has a highly practical impact. Mechanical production, however, where schedules, shift patterns, changeover times or multiple machine set-ups are of particular importance, is not an objective of this training.

Figure 2 shows the spatial organization of the *Box Game* in the learning laboratory: five work tables are arranged in a suitable location and standard positions like interim storages are marked with adhesive tape. As can be surmised, the setting can also be build up in locations such as conference rooms, training rooms, or even canteens.

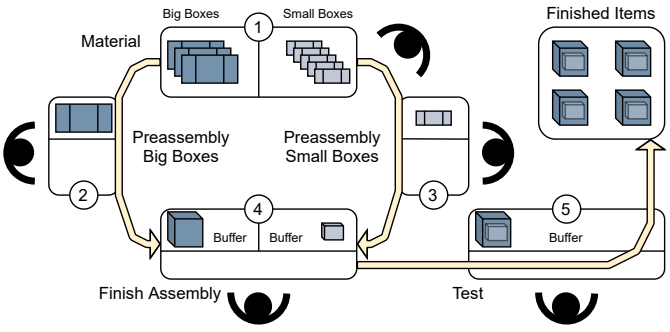

Fig. 2. Layout design of the *Box Game*.

Beside the instructor, the following can partake in the game:

5 participants who will occupy the work stations,
3 players who record the processing times,
1 observer who records inventories in the system,
1 observer who records productivity levels, and
2 further possible people who disassemble the boxes.

Though technically possible, the working stations should not be filled with more than one person. Thus, larger groups need to be split. At the five stations or (transport-wise) between them, the following activities have to be conducted:

1. **Material storage** Deliver unfolded boxes.
2. **Preassemble big boxes** Fold the big box, close its lid, and pass the box on.
3. **Preassemble small boxes** Fold the small box, close its lid, and pass the box on.
4. **Finish assembly** Open the big box, insert the small one, label the small box with a post-it as "package note", close and tape big box lid up, and pass the assembled box on.
5. **Quality test** Shake the box for an acoustic quality check, apply a red dot as "passed" to the upper left corner of the box, and place the finished box in the dispatch area.

The initial stock of the *Box Game* is 75 big and 75 small boxes. However, it is not the primary aim to produce the entire demand in the shortest possible time, but to produce them according to the customer's demand (in this case one part every 15 seconds) without much inventory and with as few employees as possible.

Recall that Figure 1 shows the value stream diagram of the *Box Game*. Although the processing times can be annotated in the diagram, it is hardly simulated due to a lacking mathematical foundation. To the authors' knowledge, even software that replicates human intuition of value stream diagrams does not exist. Nonetheless, the diagram helps to understand how the *Box Game* is played in detail.

To experience the challenges that management faces with regard to a possible strategic realignment, the box game is typically meant to be played in four rounds of 5 or 8 minutes each. During the simulation, two types of production principles with two batch sizes are examined.

Batch size 3 - push principle: The products are passed on in batches of size 3. Each process step works functionally independent from the other and the participants are rewarded for the amount of pieces they work on. Hence, it is the goal at each station to produce as much output as possible.

Batch size 3 - pull principle: Stations produce and pass on products in batches of size 3. Upstream stations have to hold their pieces and stop production until it is demanded by an internal or external customer. The capacity of a station and its buffer is limited to 3 items and items can only be replaced accordingly.

Batch size 1 - pull principle: The third round is played like the second one, but the batch size is reduced to one.

Improvement - pull principle: The last round is used to find improvements autonomously and to apply them as a team.

The advantage of this approach is that the participants gather personal experiences. This can hardly be replaced by a computer simulation. Yet, augmenting this hands-on experience by such a simulation helps to scale up both range and complexity of the considered process. This is partly due to the students' attention levels diminishing after about 5 minutes of play.

V. Basic Considerations for an IT-based Simulation

The authors' decades of experience with Petri nets and the availability of the *P-S.C* (which is freely usable for academic purposes) resulted in the decision to implement the described scenarios as Petri nets. However, this task turned out to be more challenging than assumed beforehand [34]:

1. Models and simulations of pull processes must distinguish between different customer orders. This can be expressed in high-level Petri nets with individual tokens. But although such Petri net classes have been known for many years (e. g., [9], [10]), there are no modeling patterns that can be used to build models.
   This makes testing models step by step as they are created even more important. Thus, new modeling techniques can be developed incidentally. The experience gained is therefore as much an artifact in the sense of [28] as the *P-S.C* or the models themselves.
2. Without a suitable tool for modeling and simulating high-level nets, however, this experience cannot be gained, which is a hurdle for many modelers. Almost all Petri net tools listed in [35] are either outdated, do not support time aspects or Petri nets with individual tokens, and none of them have a modern user interface. Therefore, they are all but unusable for the task at hand.
   The *P-S.C*, however, is well suited to be used for modeling the described logistics laboratory.

Section II already examined related work in order to rule out that only the authors' personal preferences justified the chosen modeling approach.

VI. Clock Pulse Simulation Models

Visualization-wise, the *P-S.C* draws nodes in a manner that allows label and token quantity to be presented inside each node. This is in contrast to the standard circles and squares (or the original lines) but allows for easier interpretation as such labels are often found in other modeling systems.

Clock pulse models evaluate every discrete time step and, thus, can be used to study the change of the systems' states. Concretely, the presented models in this section allow for the observation of fluctuating storage utilization.

Such an observation necessitates a constant flow of time. To this end, the first step in modeling the *Box Game* is the implementation of a clock that ticks every second, as depicted in Figure 3 (a). A tiny Petri net consists of a time-typed place *clock*, a corresponding transition *pulse* and two anti-parallel arcs. One arc (bearing the label *s*) removes a time token from the *clock*, while the other one adds one second to the received value and puts a token back on the *clock*. Thus, each firing increments the elapsed total time by one second.

The *P-S.C* allows for computations to be made on arcs instead of only accessing token values as is the case for the label *s*. However, when such calculations become too complex, they also become increasingly unwieldy and obstruct much space in the net's visualization.

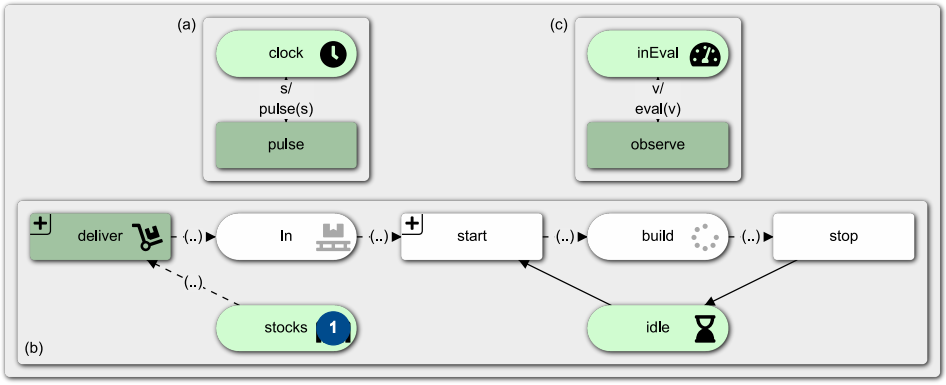

Fig. 3. Modeling concepts: (a) simple Petri net clock · (b) preassembling as instance of a single working station · (c) observer net that evaluates the upstream storage in (b).

Functions can be implemented in the net's specification and then be called from every arc - provided the passed parameters are in order. The functions' return values determine the contents of the tokens to be put on the post set's places. In case of *pulse(s)*, the function returns *s.sec+"00:01"*.

The basic model of a working station (here, folding of the big boxes as shown in Figure 3 (b)) consists of one place *in* as a buffer, one for the upstream *stocks* and one for the workplace *build* itself. These places are typed as a user-defined record set consisting of an integer as *id*, a character string as *type* descriptor and a time-value as *inStamp*, denoting the time the corresponding token was put on the place.

The transition *deliver* provides the inbound buffer *in* with feedstock from the *stocks* (and possibly other associated data) while the transition *start* connects this buffer with the workplace *build*. Both transitions carry a *select* criterion to choose the item with the minimal *id* according to the FIFO principle and put it on the post set - other queueing principles could be implemented just as well. The transition *stop* carries a *condition* to wait for the item to be finished. This *condition* is implemented as difference between the current times of the *clock* and the *inStamp* as encoded on the token. The used times are based on students' experience. Conditions and selections can be displayed in the model by clicking on the plus-symbol.

The transition *stop* is attached to an interim storage (as shown later) that serves as buffer for the succeeding working station. The (non-typed) place *idle* serves as a semaphore that prohibits *start* from firing while the workplace is busy: Only one box at a time can be processed. In the full model, it is also used as a time-typed note for the workplace.

Though single transitions can be fired by clicking on them, the *P-S.C* supports simultaneous firing of all enabled ones. This is used to synchronize all transitions with the *clock*.

An observer net, as exemplarily shown in Figure 3 (c), evaluates the costs associated with the storage. Each clock pulse, the (integer) value of the place *inEval* is increased by the amount of tokens on place *in*, thus indicating this storage's costs in this second. Again, this is implemented as a combination of a token access *v* and a function call *eval(v)*.

Combined, these three models allow for the observation of fluctuating storage levels, of bottlenecks, and of storage cost.

In the initial marking of the working station net in Figure 3 (b), there is one token each on the places *stocks* and *idle* which is indicated by their light green color. Additionally, *stocks* shows one big blue circle containing the amount of actual tokens on the place. This visual is omitted on *idle* as this semaphore may not contain more than one token (as is also the case for *clock* and *inEval*). The transition *deliver* is colored in a darker green to show it is enabled, i. e., it is ready to fire.

Figure 4 shows the remaining reachable states of the working station net. The transition *deliver* chooses the (only) token and puts it on *in*, enabling *start* as shown in Figure 4 (a). The next pulse removes the semaphore token and the one from *in*, preventing *start* from becoming enabled. This is reflected in Figure 4 (b). The marking stays stable until the *condition* on *stop* regarding the processing time evaluates to true.

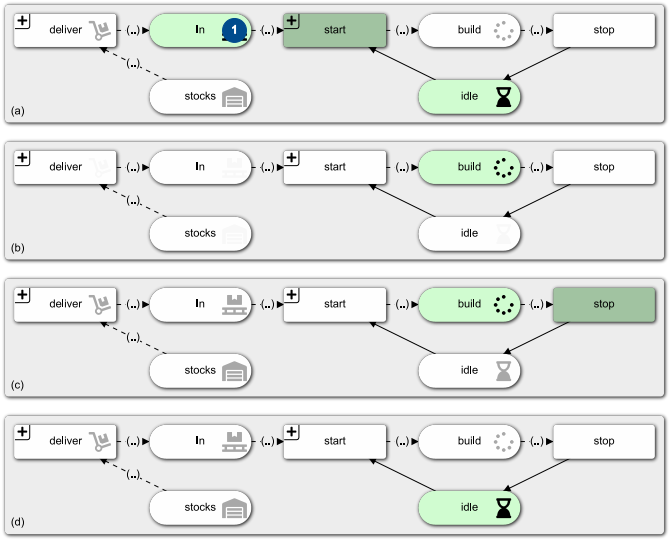

Fig. 4. Visualized states of the working station net in Figure 3 (b): (a) after clock pulse one · (b) during the building phase · (c) after clock pulse twenty · (d) after clock pulse twenty-one.

Now, the transition becomes enabled, leading to the net as shown in Figure 4 (c). Figure 4 (d) depicts the net after the item has been taken from *build* and the semaphore token has returned, rendering the working place accessible again.

TABLE I. MARKINGS OF THE MODEL AS DEPICTED IN FIGURE 3

			material			in			build		
clock	inEval	idle	id	type	inStamp	id	item	inStamp	id	type	inStamp
00:00	0	•	1	big box	00:00						
00:01	0	•				1	big box	00:01			
00:02	1								1	big box	00:02
⋮	⋮								⋮	⋮	⋮
00:20	1								1	big box	00:02
00:21	1	•									

These state changes are reflected in Table I. The first row shows the net's initial marking. As each pulse evaluates to one second, the box arrives at *build* after two seconds. There, the box is processed for 18 seconds, then leaving for the next buffer. The net reaches its final marking after 21 seconds.

All models presented here, being implemented for automated simulation, do not need any user interaction. In effect, each state has a unambiguously defined subsequent state.

A. The Push Model

By expanding on the presented working station model and using the mentioned concepts, the first iteration of the *Box Game* (the push version as presented during a simulation run in Figure 5) can be modeled. As the principle of the net can be used for the other workplaces, they can be structurally copied and then adapted with respect to the processing time.

Connected to the upstream buffer places *inBB* and *inSB* of the big and small box assembly nets, there is the already mentioned place *material* for the main warehouse. The warehouse is connected via the two delivery transitions *deliverSB* and *deliverBB*. As all these places carry item representations for tokens, they are typed with the user-defined record set *RStock* that was introduced earlier. *RStock* consists of an integer *id*, a character as *type* description and a time-value *inStamp* for the moment the token was put on the place.

Initially, *material* bears 75 tokens each for unfolded big and small boxes. The structure of this allocation can be seen in Table II. *BB* and *SB* are short for big and small box.

TABLE II. INITIAL ALLOCATION OF TOKENS ON THE PLACE *material* IN THE CLOCK PULSE PUSH MODEL AS DEPICTED IN FIGURE 5

id	type	inStamp
1	BB	00:00
⋮	⋮	⋮
75	BB	00:00
76	SB	00:00
⋮	⋮	⋮
150	SB	00:00

Both sides are structured equally, so it is sufficient to only explain one path in detail. The transition *deliverBB* carries several criteria. First, *conditions* ensure that only correct items (i. e., items designated *BB*) are chosen and that transportation has a duration of 2 seconds (i. e., *deliverBB* is only enabled if 2 seconds have elapsed since the last transport). Second, the *select* criterion chooses the item with the minimal *id*.

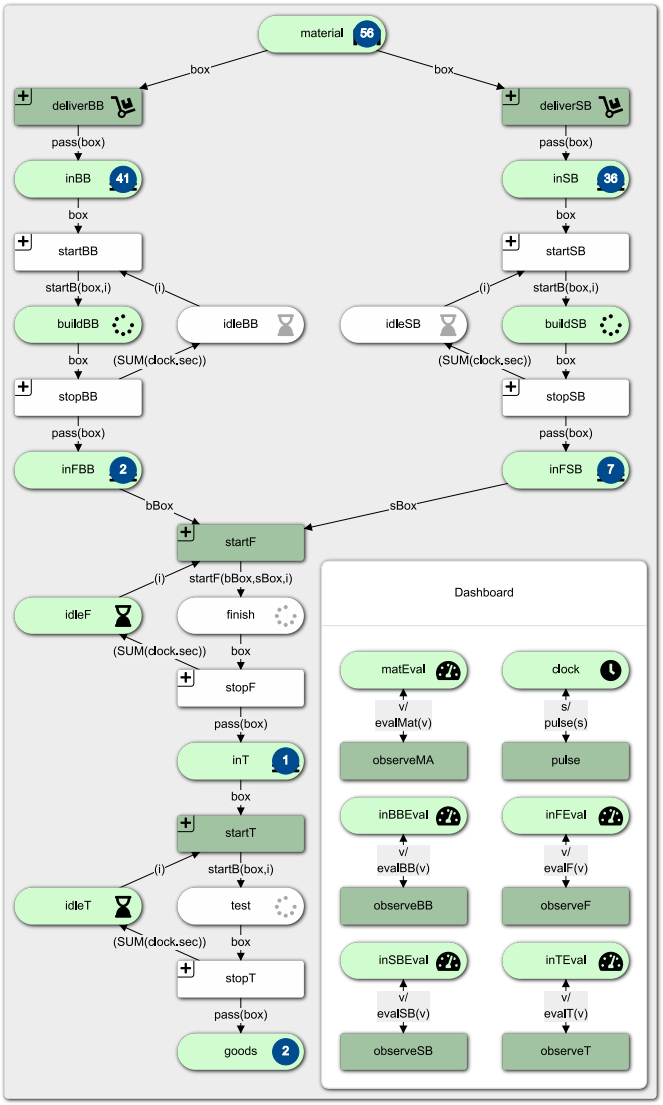

Fig. 5. Clock pulse push model after 96 clock pulses.

While the arc *box* takes one token from *material*, the function *pass(box)* sets a new timestamp (according to the elapsed time as provided by the *clock*) and puts it on *inBB*.

The transition *startBB* is enabled if there is at least one token on *inBB* and the semaphore token is available. As is the case for all semaphores, the initial token bears *"00:00"* as time value. Again, *select* criterion on *startBB* is the minimal *id*. The transition receives a *box* and the semaphore token as input data. Upon firing, the function *startB(box,i)* determines the correct putting time and places the box on *buildBB*.

Due to folding the big boxes taking 18 seconds, *stopBB* only becomes enabled after this time has elapsed.

Note the difference between the (left) big box and the (right) small box sides: Folding a small box only takes 9 seconds. Thus, the *condition* on *stopSB* adheres to this value.

Upon firing, *stopBB* puts the folded *box* on the output buffer *inFBB* that is named for its second purpose as input buffer for the following subnet to finish the assembly. Additionally, the semaphore token is put back on *idleBB*. The semaphores can be used to determine machine processing (or idle) times.

The subnet for the final assembly differs in possessing a second input buffer *inFSB* as both boxes are needed. This also causes a slightly adapted function *startF(bBox,sBox,i)* because the time-values of both incoming boxes must be considered. Structurally fully equivalent to the first two subnets is the last one for quality testing.

Like for conditions, functions and data accesses on arcs can be shown or hidden, as needed. To further enhance the visual understanding of their functionality, nodes can be provided with (animated) symbols.

Tables III to IX are meant to examine the markings' progression of the model as depicted in Figure 5.

The first row of Table III is a shorter version of Table II: Before the initial pulse (i. e., "after pulse 0") the cell *id* contains the full column *id* from Table II, the single elements being separated by the pipe symbol. The same applies to the cells *type* and *inStamp* of the first row. As we assume a transportation time of 2 seconds from *material* to the buffers, the first change occurs after pulse 2: The items *(1, BB, 00:00)* and *(76, SB, 00:00)* are consumed from *material* and put on their respective buffers. This is repeatedly shown for the first 10 pulses. The row for pulse 96 shows the tokens on *material* for the system's state in Figure 5. Pulses 150 and 151 contain the allocations just before the last consumption from the place and the final state of *material*.

As this concept also extends to all following tables, every row in them can be expanded to an own table from which the model's state after the corresponding pulse can be explored. The combination of all these tables into one huge table is the result export from the *P-S.C.* In case of the clock pulse push model, this table contains 44 columns with 1904 data sets as rows each of which corresponds to one system state. Thus, this export comprises the reachability set of the net.

On the other end of the model, the place *goods* contains the finished boxes. Initially, this place is empty. In accordance with Table IV, the first token creation takes place at pulse 53, designating the first finished product *(1, FB, 00:52)*. Thus, after 52 seconds, the first box is ready and put in the finished goods warehouse. During the next 10 pulses, no change occurs: Only after pulse 77 will the next item be ready. Pulse 96, again, shows the allocation of tokens on *goods* at the state of the model as shown in Figure 5: Two items are finished, the second one being put there after 1:17 minutes. The last token creation takes place at pulse 1903: the item *(75, FB, 31:42)* is finalized after more than half an hour. Then, the allocation does not change any more, indicating the end of the simulation run.

TABLE III. ALLOCATIONS OF TOKENS ON THE PLACE *material* IN THE CLOCK PULSE PUSH MODEL AS DEPICTED IN FIGURE 5: INITIALLY, AFTER THE FIRST 10 PULSES, AFTER PULSE 96, AND AROUND THE LAST CONSUMPTION

	material		
pulse	id	type	inStamp
0	1|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
1	1|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
2	1|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
3	2|...|75|77|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
4	2|...|75|77|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
5	3|...|75|78|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
6	3|...|75|78|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
7	4|...|75|79|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
8	4|...|75|79|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
9	5|...|75|80|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
10	5|...|75|80|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
⋮	⋮	⋮	⋮
96	48|...|75|123|...|150	BB|...|BB|SB|...|SB	00:00|...|00:00
⋮	⋮	⋮	⋮
150	75|150	BB|SB	00:00|00:00
151			

TABLE IV. ALLOCATIONS OF TOKENS ON THE PLACE *goods* IN THE CLOCK PULSE PUSH MODEL AS DEPICTED IN FIGURE 5: AT FIRST TOKEN CREATION AND THE FOLLOWING 10 PULSES, AFTER PULSE 96, AND AROUND THE LAST TOKEN CREATION

	goods		
pulse	id	type	inStamp
53	1	FB	00:52
54	1	FB	00:52
55	1	FB	00:52
56	1	FB	00:52
57	1	FB	00:52
58	1	FB	00:52
59	1	FB	00:52
60	1	FB	00:52
61	1	FB	00:52
62	1	FB	00:52
63	1	FB	00:52
⋮	⋮	⋮	⋮
96	1|2	FB|FB	00:52|01:17
⋮	⋮	⋮	⋮
1903	1|...|75	FB|...|FB	00:52|...|31:42
⋮	⋮	⋮	⋮

Between *material* and *goods*, a lot is happening that can be deducted from the different tables.

Table V shows data for the inbound buffers *inBB* and *inSB*. These places are initially empty. The first creation of tokens occurs after transportation from *material*, thus, the boxes are put with an *inStamp* of 2 seconds. Both boxes can be transferred into the preassembly and, therefore, disappear with the next pulse. After this, the buffer gets filled, because transportation takes less time than folding. Again, pulse 96 shows the allocation for the shown model in Figure 5. Due to folding of the small boxes being faster in comparison to the big boxes, the last consumption from *inSB* takes place at pulse 670, while the last one from *inBB* happens at pulse 1336.

TABLE V. ALLOCATIONS OF TOKENS ON THE PLACES *inBB* AND *inSB* IN THE CLOCK PULSE PUSH MODEL AS DEPICTED IN FIGURE 5: AT FIRST TOKEN CREATION AND THE FOLLOWING 10 PULSES, AFTER PULSE 96, AND AROUND THE LAST CONSUMPTION FROM THE PLACES

	inBB			inSB		
pulse	id	type	inStamp	id	type	inStamp
3	1	BB	00:02	76	SB	00:02
4						
5	2	BB	00:04	77	SB	00:04
6	2	BB	00:04	77	SB	00:04
7	2|3	BB|BB	00:04|00:06	77|78	SB|SB	00:04|00:06
8	2|3	BB|BB	00:04|00:06	77|78	SB|SB	00:04|00:06
9	2|3|4	BB|BB|BB	00:04|00:06|00:08	77|78|79	SB|SB|SB	00:04|00:06|00:08
10	2|3|4	BB|BB|BB	00:04|00:06|00:08	77|78|79	SB|SB|SB	00:04|00:06|00:08
11	2|3|4|5	BB|BB|BB|BB	00:04|00:06|00:08|00:10	77|78|79|80	SB|SB|SB|SB	00:04|00:06|00:08|00:10
12	2|3|4|5	BB|BB|BB|BB	00:04|00:06|00:08|00:10	77|78|79|80	SB|SB|SB|SB	00:04|00:06|00:08|00:10
13	2|...|6	BB|...|BB	00:04|...|00:12	77|...|81	SB|...|SB	00:04|...|00:12
⋮	⋮	⋮	⋮	⋮	⋮	⋮
96	7|...|47	BB|...|BB	00:14|...|01:34	87|...|122	SB|...|SB	00:24|...|01:34
⋮	⋮	⋮	⋮	⋮	⋮	⋮
669	38|...|75	BB|...|BB	01:16|...|02:30	150	SB	02:30
670	39|...|75	BB|...|BB	01:18|...|02:30			
⋮	⋮	⋮	⋮			
1335	75	BB	02:30			
1336						

TABLE VI. ALLOCATIONS OF TOKENS ON THE PLACES *buildBB* AND *buildSB* IN THE CLOCK PULSE PUSH MODEL AS DEPICTED IN FIGURE 5: AT FIRST TOKEN CREATION AND THE FOLLOWING 10 PULSES, AFTER PULSE 96, AND AROUND THE LAST CONSUMPTION FROM THE PLACES

	buildBB			buildSB		
pulse	id	type	inStamp	id	type	inStamp
4	1	BB	00:02	76	SB	00:02
5	1	BB	00:02	76	SB	00:02
6	1	BB	00:02	76	SB	00:02
7	1	BB	00:02	76	SB	00:02
8	1	BB	00:02	76	SB	00:02
9	1	BB	00:02	76	SB	00:02
10	1	BB	00:02	76	SB	00:02
11	1	BB	00:02	76	SB	00:02
12	1	BB	00:02			
13	1	BB	00:02	77	SB	00:11
14	1	BB	00:02	77	SB	00:11
⋮	⋮	⋮	⋮	⋮	⋮	⋮
96	6	BB	01:32	86	SB	01:32
⋮	⋮	⋮	⋮	⋮	⋮	⋮
677	38	BB	11:08	150	SB	11:08
678	38	BB	11:08			
⋮	⋮	⋮	⋮			
1352	75	BB	22:14			
1353						

Afterwards, the boxes are taken from the buffers directly into the first production steps. Note the assumption that this handling step consumes no time, i. e., the placing of a box into a buffer allows for immediate start of the production step (given the working place is available). This is due to the *Box Game*' layout: tables are in close vicinity to each other and buffers are directly connected to their working stations.

Handling times could be implemented like it is the case for the initial transportation between *material* and the first buffers.

As shown in Table VI, although being in pulse 4, *inStamp* reads *00:02* for the first preassembly. Folding a small box takes 9 seconds. After pulse 11 this time is reached and the folded small box can be transferred to the next buffer *inFSB*. Again, note the *inStamp* of the next small box being *00:11* due to the absence of handling times. The big box is not folded until pulse 20. After pulse 677, the last consumption from *buildSB* takes place, while pulse 1352 marks the last consumption from *buildBB*. Then, both places remain empty.

Table VII (a) shows the buffers behind *buildBB* and *buildSB* where the folded boxes are temporarily stored. As before, handling or transportation times are not considered, thus the *inStamp* of small box *76* reads *00:11*. The first folded big box *1* arrives at the buffer only at pulse 21. In the next pulse, the first available tokens get consumed, effectively transferring the boxes to the final assembly. Once more, pulse 96 depicts the allocation of tokens in the model's state from Figure 5 while the last two rows show the consumption of the last tokens. The small box *150* shows some 11 minutes more waiting time compared to the big box *75*.

Starting with pulse 22, a big and a small box each are transferred from *inFBB* and *inFSB* to the final assembly. Here, the small box and a note are put into the larger. The construct gets a seal and is passed on. Finishing the box takes time up to pulse 45, thus being not reflected in Table VII (b), although the the principle stays the same. At pulse 96, *finish* is not marked because (as elaborated shortly) a finished box is in transit to the next buffer and the two following boxes are in transit to the final assembly. Pulse 1895 marks the consumption of the last finished box.

TABLE VII. ALLOCATIONS OF TOKENS ON THE PLACES (a) *inFBB*, *inFSB* AND (b) *finish* IN THE CLOCK PULSE PUSH MODEL AS DEPICTED IN FIGURE 5: AT FIRST TOKEN CREATION AND THE FOLLOWING 10 PULSES, AFTER PULSE 96, AND AROUND THE LAST CONSUMPTION FROM THE PLACES

(a)	inFBB			inFSB		
pulse	id	type	inStamp	id	type	inStamp
12				76	SB	00:11
13				76	SB	00:11
14				76	SB	00:11
15				76	SB	00:11
16				76	SB	00:11
17				76	SB	00:11
18				76	SB	00:11
19				76	SB	00:11
20				76	SB	00:11
21	1	BB	00:20	76|77	SB|SB	00:11|00:20
22				77	SB	00:20
⋮	⋮	⋮	⋮	⋮	⋮	⋮
96	4|5	BB|BB	01:14|01:32	79|...|85	SB|...|SB	00:38|...|01:32
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1871	75	BB	22:32	150	SB	11:17
1872						

(b)	finish		
pulse	id	type	inStamp
22	1	FB	00:20
23	1	FB	00:20
24	1	FB	00:20
25	1	FB	00:20
26	1	FB	00:20
27	1	FB	00:20
28	1	FB	00:20
29	1	FB	00:20
30	1	FB	00:20
31	1	FB	00:20
32	1	FB	00:20
⋮	⋮	⋮	⋮
96			
⋮	⋮	⋮	⋮
1895	75	FB	31:10
1896			

TABLE VIII. ALLOCATIONS OF TOKENS ON THE PLACES (a) *inT* AND (b) *test* IN THE CLOCK PULSE PUSH MODEL AS DEPICTED IN FIGURE 5: AT FIRST TOKEN CREATION AND THE FOLLOWING 10 PULSES, AFTER PULSE 96, AND AROUND THE LAST CONSUMPTION FROM THE PLACES

(a)	inT		
pulse	id	type	inStamp
46	1	FB	00:45
47			
48			
49			
50			
51			
52			
53			
54			
55			
56			
⋮	⋮	⋮	⋮
96	3	FB	01:35
⋮	⋮	⋮	⋮
1896	75	FB	31:35
1897			

(b)	test		
pulse	id	type	inStamp
47	1	FB	00:45
48	1	FB	00:45
49	1	FB	00:45
50	1	FB	00:45
51	1	FB	00:45
52	1	FB	00:45
53			
54			
55			
56			
57			
⋮	⋮	⋮	⋮
96			
⋮	⋮	⋮	⋮
1902	75	FB	31:35
1903			

TABLE IX. ALLOCATIONS OF TOKENS ON THE PLACES (a) *matEval*, *inBBEval*, *inSBEval*, *inFEval*, AND (b) *inTEval* INDICATING THE ACCUMULATED STORAGE COSTS IN THE CLOCK PULSE PUSH MODEL AS DEPICTED IN FIGURE 5: INITIALLY, AFTER THE FIRST 10 PULSES, AROUND PULSE 96, AND AT THE LAST TOKEN CREATING PULSE

pulse	matEval	inBBEval	inSBEval	inFEval	inTEval
0	0	0	0	0	0
1	150	0	0	0	0
2	300	0	0	0	0
3	450	0	0	0	0
4	598	1	1	0	0
5	746	1	1	0	0
6	892	2	2	0	0
7	1038	3	3	0	0
8	1182	5	5	0	0
9	1326	7	7	0	0
10	1468	10	10	0	0
⋮	⋮	⋮	⋮	⋮	⋮
95	9926	1886	1656	327	2
96	9982	1927	1692	336	2
97	10038	1968	1728	345	3
⋮	⋮	⋮	⋮	⋮	⋮
1903	11550	44475	19500	64650	75

While Table VIII (a) examines the behavior for the last buffer *inT* just like it is the case for the other corresponding buffers, Table VIII (b) does so for the last working station *test* in accordance to the others.

The dashboard view shows the already explained *clock*: Initialized with *"00:00"*, each tick adds one second to the token. All evaluation places $*Eval$ are initialized with *0*: no costs have accumulated, yet. For each pulse, these values are accessed by the arcs designated *v*. Then, the amount of tokens on the corresponding buffer gets added to the value (by means of the respective $eval*(v)$ function) which gets returned to the evaluation places. Thus, every item on any of the buffers creates one unit of costs every second on these buffers.

Table IX gives an overview of the storage costs' development. Although it would be possible to also include the storage costs of the finished *goods* warehouse, this is omitted because the goal of the models is to examine the difference of internal storage costs when switching from push to pull production. As throughput is limited by the final assembly step, taking 25 seconds, the inflow on *goods* remains the same regardless of the implemented principle.

B. The Pull Model

The pull version of the *Box Game* as depicted in Figure 6 is based on the previous push version. Some additional elements facilitate the implementation of pull principles.

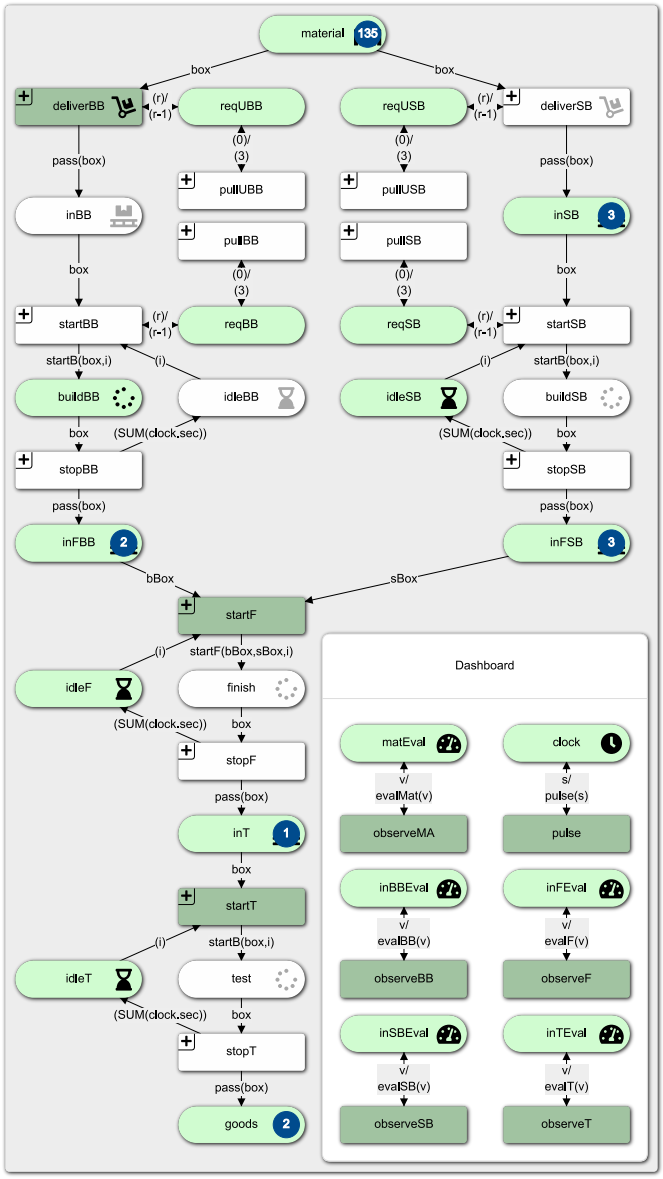

Fig. 6. Clock pulse pull model after 96 pulses.

New places *req*∗ for requesting boxes are attached to the transition for delivery and starting the preassembly. These places are integer-typed and initialized with a single token bearing *0*. The transitions *pull*∗ increases this value to 3, effectively implementing a batch size. This can be modeled with a functionality of the *P-S.C* not normally seen in Petri nets: Observance of other net elements. Exemplary, *pullUBB* carries a *condition='COUNT(inBB)=0'*: this transition observes the state of a place not directly connected. If *inBB* is empty, this triggers a kanban request. The displayed buffer *inBB* is empty, thus having enabled *pullUBB*. This led to the token on *reqUBB* containing *3*. Only then *deliverBB* is enabled as a *condition* asks for the request to be larger than *0*. The same principle triggers the start of the preassembly: *pullBB* observes the state of *inFBB*. Correspondingly, this is also executed on the small boxes' side.

While such an observance capability can be reproduced by more traditional Petri nets, the implemented approach may ease readability of the net as fewer arcs are needed. It is up to the modeler which visualization they deem more important.

Consequently, this pull request mechanism could also be implemented for the remainder of the net. However, these requests are omitted for two reasons:

1. The visualization would become more convoluted.
2. They are not necessary for this model. The final assembly *finish* takes the longest time of all working stations, thus representing the bottleneck. Before this place (including the associated buffers *inFBB* and *inFSB*), the kanban system is implemented. Behind this place, no congestion can occur. As only storage costs are examined, and the storage costs are the same for all buffers, total costs would not change when implementing a full kanban chain.

The remaining net is unchanged to the push version, thus, only the main difference on the resulting reachability set are examined: Indications for storage costs. As such, Table X shows the data for the pull model like Table IX does for the push model. While *material* exposes higher costs, the internal storage costs for the buffer prove smaller than in the push model. This is more deeply explored in Section VI-C.

TABLE X. ALLOCATIONS OF TOKENS ON THE PLACES (a) *matEval*, *inBBEval*, *inSBEval*, *inFEval*, AND (b) *inTEval* INDICATING THE ACCUMULATED STORAGE COSTS IN THE CLOCK PULSE PULL MODEL AS DEPICTED IN FIGURE 6: INITIALLY, AFTER THE FIRST 10 PULSES, AROUND PULSE 96, AND AT THE LAST TOKEN CREATING PULSE

pulse	matEval	inBBEval	inSBEval	inFEval	inTEval
0	0	0	0	0	0
1	150	0	0	0	0
2	300	0	0	0	0
3	450	0	0	0	0
4	598	1	1	0	0
5	746	1	1	0	0
6	892	2	2	0	0
7	1038	3	3	0	0
8	1182	5	5	0	0
9	1326	7	7	0	0
10	1470	9	9	0	0
⋮	⋮	⋮	⋮	⋮	⋮
95	13338	144	196	157	2
96	13473	144	199	162	2
97	13608	144	202	167	3
⋮	⋮	⋮	⋮	⋮	⋮
1903	127068	4133	4584	4926	75

C. Simulation Results

Comparison of the two models' simulation runs show neither a change in total processing time nor one in idle times for the different workplaces, which is as expected. Differences on utilization of storage places become visible, though. Figures 5 and 6 depict both clock pulse models after pulse 96. Thus, they are in a comparable state but clearly show distinct distributions on the earlier buffers. This is even more evident when comparing the last rows of Tables IX and X.

The first model using push principles clearly unveils the drawback of its approach: large interim storage and, as a result, high inventory costs. Figure 7 (upper) depicts the (box-wise split) inventory on the material storage (*rawBB* and *rawSB*), the building buffers (*inBB*, *inSB*, *inFBB*, and *inFSB*) and the finished *goods* storage during the push simulation.

Figure 7 (lower) shows the pull simulation run's results. The interim storage places are much less utilized as only those items are put into the assembly lines that are demanded by downstream stations. The decrease in raw material stocks is very uniform. Both is as expected, as it corresponds to the main goal behind just-in-time production schedules.

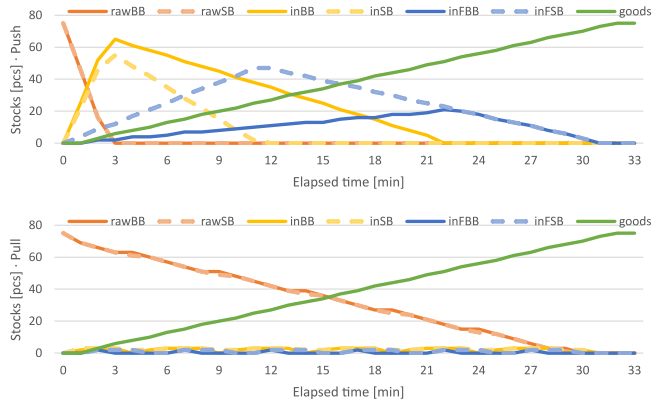

Fig. 7. Stocks in the clock pulse models for (upper) push and (lower) pull.

Figure 8 (upper) presents the accumulated costs of interim storage in the push model, while Figure 8 (lower) depicts the same for the pull model. Also shown are the total costs of all interim storage places. Be mindful of the scale difference of the y-axis: The accumulated interim storage costs differ by an order of magnitude when using pull instead of push principles!

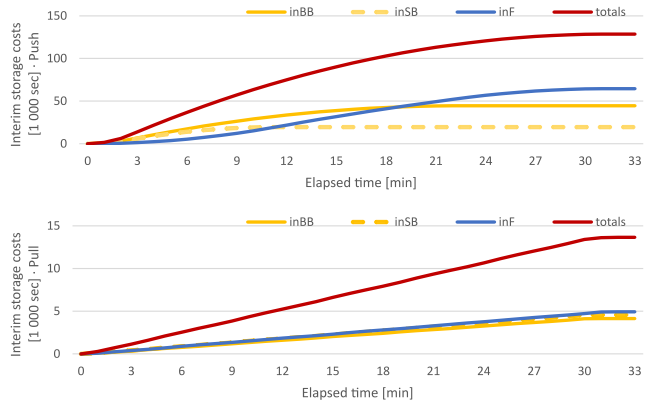

Fig. 8. Accumulated inventory costs per interim storage and totals in the clock pulse models for (upper) push and (lower) pull.

The finished *goods* storage is omitted as throughput is the same in both models, leading to the same costs on this storage. The *material* warehouse, otherwise, is put aside because the difference between the accumulated totals equals to the possible savings when externalizing this storage.

By implementing pull principles, the stock costs on the interim storage places plummet. Thus, the advantages of a just-in-time production and a smaller main storage become obvious. The pull principle allows for exactly this.

What is not accounted for in these models are for example costs of transportation, as smaller batch sizes usually correspond to higher transportation costs, leading to familiar knowledge: decreasing one muda typically increases other muda. Hence, batch size 1, which is optimal for the interim storage cost, is not necessarily the globally optimal solution.

VII. Event Triggered Simulation Models

The second modeling approach examined aims at developing an event triggered simulation model, again with the main goal to determine the total costs of all involved stocks and to explain the applied modeling technique. Thus, the setting remains the same but the modeling principles change. While the clock pulse models allow for observation of storage utilization in real-time, event triggered models aim at computing simulation results as fast as possible.

Hence, the now presented Petri net models mainly consist of places to examine the stocks. The remainder of the *Box Game*'s functionality is implemented as transitions and arcs. Specifically, there are no places for the working stations.

Since an ideal production flow is one with minimal stocks and short throughput times, the individual boxes are passed from one production step directly to the following, effectively establishing a batch size of 1. This is in accordance with the handling times of the clock pulse models that were set to *0* with the exception of the initial dispatch. Other batch sizes are implemented accordingly [5].

A. The Push Model

Figure 9 shows the push version of the event triggered model. The *timer* establishes one possibility to track elapsed times (another one is presented in the following Section VII-B). The initial allocation with four tokens can be seen in Table XI. The used record set *RTimer* stores the *type*, i. e., the associated storage place, and the corresponding *elapsed* time value of last access on this place. The tokens are initialized with *"00:00"* the buffers have not been in use, yet.

TABLE XI. INITIAL ALLOCATION OF TOKENS ON THE PLACE *timer* IN THE EVENT TRIGGERED PUSH MODEL AS DEPICTED IN FIGURE 9

type	elapsed
BB	00:00
SB	00:00
FB	00:00
QA	00:00

The place *material* is typed with a record set *RMat* that consists of the item *type*, the *elapsed* time, and two attributes *boxID* and *count*. While *boxID* determines the next box's id value, *count* shows the amount of the remaining stock. The attribute *type* is self-explanatory and *elapsed* tracks the time of an item's state change.

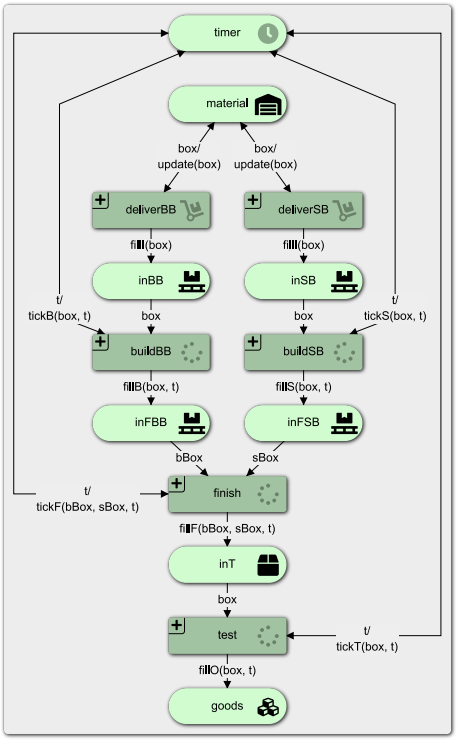

Fig. 9. Event triggered push model after 4 ticks, the first completed box having just arrived on the finished *goods* warehouse.

The initial allocation for *material* is shown in Table XII: There are 75 big and small boxes, respectively. Both (big and small box) tokens that are the next to be created from this information will have the *id 1*.

TABLE XII. INITIAL ALLOCATION OF TOKENS ON THE PLACE *material* IN THE EVENT TRIGGERED PUSH MODEL AS DEPICTED IN FIGURE 9

boxID	type	elapsed	count
1	BB	00:00	75
1	SB	00:00	75

Left and right sides of the preassembly work correspondingly, thus, an explanation of the big boxes' side will suffice. Two *conditions* on *deliverBB* ensure usage of the *BB*-typed token from *material* while there are items left as indicated by the token's *count*. The information gets adapted by the function *update(box)*: *boxID* is incremented, *count* is decremented, and *elapsed* receives a time stamp for the last access. Then, the token is put back on *material* and a newly created *RBox*-typed token is put on *inBB*.

With creation of a box token, *boxID* actually becomes the box's *id* while *count* as attribute on a box becomes superfluous. Thus, *RBox* is adapted as a triplet *id*, *type*, and *elapsed*. Using the *id*, the items can be tracked in this model's tables.

The progression of token allocations on *timer*, *material* warehouse, and the buffers *inBB* and *inSB* can excerpted be seen in Tables XIII, XIV, and XV, respectively.

TABLE XIII. ALLOCATIONS OF TOKENS ON THE PLACE *timer* IN THE EVENT TRIGGERED PUSH MODEL AS DEPICTED IN FIGURE 6 FOR THE FIRST AND LAST FIVE SYSTEM STATES EACH

state	type	elapsed
0	BB|SB|FB|QA	00:00|00:00|00:00|00:00
1	BB|SB|FB|QA	00:00|00:00|00:00|00:00
2	BB|SB|FB|QA	00:20|00:11|00:00|00:00
3	BB|SB|FB|QA	00:38|00:20|00:45|00:00
4	BB|SB|FB|QA	00:56|00:29|01:10|00:52
⋮	⋮	⋮
74	BB|SB|FB|QA	21:56|10:59|30:20|30:02
75	BB|SB|FB|QA	22:14|11:08|30:45|30:27
76	BB|SB|FB|QA	22:32|11:17|31:10|30:52
77	BB|SB|FB|QA	22:32|11:17|31:35|31:17
78	BB|SB|FB|QA	22:32|11:17|31:35|31:42

TABLE XIV. ALLOCATIONS OF TOKENS ON THE PLACE *material* IN THE EVENT TRIGGERED PUSH MODEL AS DEPICTED IN FIGURE 6 FOR THE FIRST AND LAST FIVE SYSTEM STATES EACH

state	boxID	type	elapsed	count
0	1|1	BB|SB	00:00|00:00	75|75
1	2|2	BB|SB	00:00|00:00	74|74
2	3|3	BB|SB	00:00|00:00	73|73
3	4|4	BB|SB	00:00|00:00	72|72
4	5|5	BB|SB	00:00|00:00	71|71
⋮	⋮	⋮	⋮	⋮
74	75|75	BB|SB	02:28|02:28	1|1
75	76|76	BB|SB	02:30|02:30	0|0
76	76|76	BB|SB	02:30|02:30	0|0
77	76|76	BB|SB	02:30|02:30	0|0
78	76|76	BB|SB	02:30|02:30	0|0

TABLE XV. ALLOCATIONS OF TOKENS ON THE PLACES *inBB* AND *inSB* IN THE EVENT TRIGGERED PUSH MODEL AS DEPICTED IN FIGURE 6 FOR THE FIRST AND LAST FIVE SYSTEM STATES EACH

	inBB			inSB		
state	id	type	elapsed	id	type	elapsed
0						
1	1	BB	00:02	1	SB	00:02
2	2	BB	00:04	2	SB	00:04
3	3	BB	00:06	3	SB	00:06
4	4	BB	00:08	4	SB	00:08
⋮	⋮	⋮	⋮	⋮	⋮	⋮
74	74	BB	02:28	74	SB	02:28
75	75	BB	02:30	75	SB	02:30
76						
77						
78						

After the token has arrived on *inBB*, *buildBB* is enabled. It receives tokens from *inBB* and *timer* and transmits tokens to *inFBB* and back to *timer*. The token from *timer* is updated by the function *tick(box,t)* such that *t.elapsed=max(box.elapsed,t.elapsed)+"00:18"*: To the current time, as computed by the maximum function, 18 seconds are added for the preassembly. Thus, the *timer* token always carries the time of last access on. In the system's base state, this token is initialized with *"00:00"* as the workplaces have not been accessed, yet. However, because the delivery takes 2 seconds, the folding must not start earlier. Again, the further transportation steps are assumed to consume no time.

Using a generic function with a fitting parameter for all arcs instead of a unique one for each would have led to larger arc labels, convoluting the visualization. On the other hand, the processing times would have been visible directly.

The function *fillB(box,t)* adapts the principle behind *tickB(box,t)* to a box item where the attribute *elapsed* stores the time of last change. Allocations of tokens on the two buffers *inFBB* and *inFSB* just before the final assembly can be examined in Table XVI. The first folded big box arrives after 20 seconds. At this time, the second folded small box arrives in the other buffer, already. Du to the final assembly requiring both box types, the buffer get cleared simultaneously, though the small boxes show much longer waiting times.

TABLE XVI. ALLOCATIONS OF TOKENS ON THE PLACES *inFBB* AND *inFSB* IN THE EVENT TRIGGERED PUSH MODEL AS DEPICTED IN FIGURE 6 FOR THE FIRST AND LAST FIVE SYSTEM STATES EACH

	inFBB			inFSB		
state	id	type	elapsed	id	type	elapsed
0						
1						
2	1	BB	00:20	1	SB	00:11
3	2	BB	00:38	2	SB	00:20
4	3	BB	00:56	3	SB	00:29
⋮	⋮	⋮	⋮	⋮	⋮	⋮
74	73	BB	21:56	73	SB	10:59
75	74	BB	22:14	74	SB	11:08
76	75	BB	22:32	75	SB	11:17
77						
78						

Both *tickF(bBox,sBox,t)* and *fillF(bBox,sBox,t)* work in the same fashion, though they need both box tokens as input because the big boxes arrive later than the small ones: the maximum of three possible values (including the *timer*) determines the availability to the next processing step.

Table XVII shows the allocations on the last buffer *inT* and the finished *goods* storage. Unsurprisingly, the values match the ones from the clock pulse push model.

TABLE XVII. ALLOCATIONS OF TOKENS ON THE PLACES *inT* AND *goods* IN THE EVENT TRIGGERED PUSH MODEL AS DEPICTED IN FIGURE 6 FOR THE FIRST AND LAST FIVE SYSTEM STATES EACH

	inT			goods		
state	id	type	elapsed	id	type	elapsed
0						
1						
2						
3	1	FB	00:45			
4	2	FB	01:10	1	FB	00:52
⋮	⋮	⋮	⋮	⋮	⋮	⋮
74	72	FB	30:20	1|...|71	FB|...|FB	00:52|...|30:02
75	73	FB	30:45	1|...|72	FB|...|FB	00:52|...|30:27
76	74	FB	31:10	1|...|73	FB|...|FB	00:52|...|30:52
77	75	FB	31:35	1|...|74	FB|...|FB	00:52|...|31:17
78				1|...|75	FB|...|FB	00:52|...|31:42

Again, all these tables represent the entire reachability set of the Petri net model, although the combined table is much smaller in comparison to the clock pulse push model: It contains only 24 columns and 78 data sets or rows. However, storage costs need to be computed separately as a dashboard or tracking functionality was not implemented.

B. The Pull Model

The event triggered pull version of the *Box Game* as depicted in Figure 10 is partly more complicated, as supplementary elements are needed to implement pull requests. Otherwise, as time logging can be integrated by tracing token information in the net forward and backward, the *timer* and corresponding arcs may be dropped: instead of sending time information from and to a dedicated *timer* place, it can be integrated directly in the tokens and updated as needed.

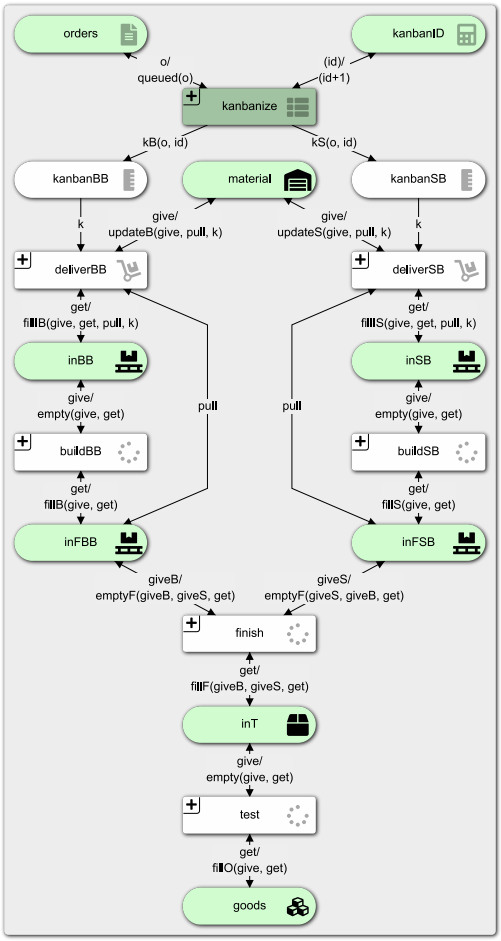

Fig. 10. Event triggered pull model after 5 ticks, the first completed box having just arrived on the finished goods warehouse.

As mentioned in Section II-A, information flows from the dispatch warehouse upstream. Thus, the kanban chain should start at *goods* instead of where it is modeled. However, kanban elements that are theoretically necessary for the model may partly be omitted for simplification.

Skipping these connections is feasible for the same reasons as in the clock pulse version: all pulls beside the modeled ones arise from working steps that require less time than *finish*, so their storage places are empty by specification at the relevant times, plus the omission eases readability of the visualization. For demonstration purposes, the batch size is set to *1*. Modeling the full kanban chain becomes necessary with larger batch sizes, i. e., when the buffers are actually filled.

The first additional elements in the event triggered pull model are the places *orders* and *kanbanID*, the latter being an integer to be used as *id* for single kanban requests. The *orders* place is typed with *ROrder*, consisting of the target *type*, the total ordered *volume*, the *batchSize*, and the amount of already *queued* items. The initial allocation of *orders* can be seen in Table XVIII. For this model, there is only one order placed for a total of 75 items. If there were more orders, an additional attribute *orderID* would become necessary to be tracked all through the net. The *kanbanID* is a sequential number initialized with *0*.

TABLE XVIII. INITIAL ALLOCATION ON THE PLACE *orders* IN THE EVENT TRIGGERED PULL MODEL AS DEPICTED IN FIGURE 10

type	volume	batchSize	queued
FB	75	1	0

Two places, *kanbanBB* and *kanbanSB*, are not initialized. They are typed with the record set *RKanban*. A token is created for each kanban request when the transition *kanbanize* fires. The *id* is the one as supplied by *kanbanID*, *type* defines the necessary intermediate product (that could be stored on a place in a bill of materials, but is hard-coded in this model), and the *batchSize* that is directly copied from the *order*.

As a first step, tokens on *orders* are transposed into kanban tokens that adhere to the given batch size. To this end, the transmission *o* supplies the order that gets updated by *queued(o)* (adding the value of *batchSize* to the current value of *queued*) and put back on its original place. The information is also used to create kanban tokens on *kanbanBB* and *kanbanSB* via the function calls *k∗(o,id)*, integrating the corresponding kanban *id*. The kanban *id* itself is incremented and also put back on its origin. Table XIX gives an overview of the token allocations on the places *orders* and *kanbanID*.

TABLE XIX. ALLOCATIONS OF TOKENS ON THE PLACES *orders* AND *kanbanID* IN THE EVENT TRIGGERED PULL MODEL AS DEPICTED IN FIGURE 10 FOR THE FIRST THREE AND AROUND THE LAST TOKEN CREATING SYSTEM STATE

	orders				kanbanID
state	type	volume	batchSize	queued	id
0	FB	75	1	0	0
1	FB	75	1	1	1
2	FB	75	1	2	2
⋮	⋮	⋮	⋮	⋮	⋮
74	FB	75	1	74	74
75	FB	75	1	75	75
76	FB	75	1	75	75

The *k∗(o,id)* functions create tokens as excerpted in Table XX. The kanban tokens slowly keep accumulating on their respective places *kanbanBB* or *kanbanSB*.

TABLE XX. ALLOCATIONS OF TOKENS ON THE PLACES *kanbanBB* AND *kanbanSB* IN THE EVENT TRIGGERED PULL MODEL AS DEPICTED IN FIGURE 10 FOR THE FIRST AND LAST FIVE SYSTEM STATES EACH WHERE THESE PLACES CARRY TOKENS

	kanbanBB			kanbanSB		
state	id	type	batchSize	id	type	batchSize
1	1	BB	1	1	SB	1
2	2	BB	1	2	SB	1
3	2|3	BB|BB	1|1	2|3	SB|SB	1|1
4	2|3|4	BB|BB|BB	1|1|1	2|3|4	SB|SB|SB	1|1|1
5	3|4|5	BB|BB|BB	1|1|1	3|4|5	SB|SB|SB	1|1|1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
219	74|75	BB|BB	1|1	74|75	SB|SB	1|1
220	74|75	BB|BB	1|1	74|75	SB|SB	1|11
221	75	BB	1	75	SB	1
222	75	BB	1	75	SB	1
223	75	BB	1	75	SB	1

As seen comparably in other models, *material* is initialized with two tokens, one for each type. However, in this model the record set is used differently: Instead of ”moving tokens through the net”, several places are equipped with tokens and information moves in the net while the tokens themselves effectively stay on their original places. The initial allocation for *material* is presented in Table XXI, also showing the structure of the record set *RMat*: item *type*, *elapsed* time denoting last access, and *count* of available stocks.

TABLE XXI. INITIAL ALLOCATION OF TOKENS ON THE PLACE *material* IN THE EVENT TRIGGERED PULL MODEL AS DEPICTED IN FIGURE 10

type	elapsed	count
BB	00:00	75
SB	00:00	75

The remaining places *inBB*, *inSB*, *inFBB*, *inFSB*, *inT*, and *goods* are typed with the record set *RBox*, which expands *RMat* by the batch size, *orderID*, and *kanbanID*. For clarity of presentation, both *orderID* and *kanbanID*, though being used for *RBox*, is not shown in any of the tables! The initialization of these places is shown in Table XXII. The *type* denotes the kind of (intermediate) product that is allowed on the corresponding place.

TABLE XXII. INITIAL ALLOCATION OF TOKENS ON THE PLACES *inBB*, *inSB*, *inFBB*, *inFSB*, *inT*, AND *goods* IN THE EVENT TRIGGERED PULL MODEL AS DEPICTED IN FIGURE 10

place	type	elapsed	count	batchSize
inBB	BB	00:00	0	0
inSB	SB	00:00	0	0
inFBB	BB	00:00	0	0
inFSB	SB	00:00	0	0
inT	FB	00:00	0	0
goods	FB	00:00	0	0

Again, big box and small box preassemblies work the same way, thus only one side is explained in detail. The delivery transition *deliverBB* is enabled when the adjacent places carry fitting tokens, i. e., when following conditions are met:

1. There must not be an item on *inFBB*, i. e., *pull.count=0*, otherwise there would not be the need for a *pull* request. Also, the *pull* should be for the item *type* the delivery transition is actually relevant for, here *pull.type="BB"*.
2. There must not be an item on *inBB*, i. e., *get.count=0* as this would effectively lead to an item on *inFBB*, again rendering the *pull* unnecessary.
3. The fitting items must be supplied by the *material* warehouse as indicated by the kanban request, i. e., *give.type=k.type*. Additionally, the corresponding stock must meet the batch size, i. e., *give.count>=k.batchSize*.

Upon firing *deliverBB*, the corresponding pull request token gets removed from *kanbanBB*. The token on *material* is updated to reflect the withdrawal of items and the access time on the place. A new token is created on *inBB* that contains information about *elapsed* time, the item *count*, and the *batchSize*. The *pull* token is transferred back to its origin without a change. The functions *update∗(give,pull,k)* and *fill∗(give,get,pull,k)* account for the necessary delivery times. Also, the functions on the right side handling the small boxes account for the shorter processing time of those boxes, leading to a later dispatch. Table XXIII gives an overview over the progression of token allocations on *material*.

TABLE XXIII. ALLOCATIONS OF TOKENS ON THE PLACE *material* IN THE EVENT TRIGGERED PULL MODEL AS DEPICTED IN FIGURE 10 AROUND THE FIRST AND LAST FIVE SYSTEM STATES WITH TOKEN CHANGES

state	type	elapsed	count
1	BB|SB	00:00|00:00	75|75
2	BB|SB	00:00|00:09	74|74
3	BB|SB	00:00|00:09	74|74
4	BB|SB	00:00|00:09	74|74
5	BB|SB	00:15|00:24	73|73
⋮	⋮	⋮	⋮
220	BB|SB	29:50|29:59	2|2
221	BB|SB	30:15|30:24	1|1
222	BB|SB	30:15|30:24	1|1
223	BB|SB	30:15|30:24	1|1
224	BB|SB	30:40|30:49	0|0
⋮	⋮	⋮	⋮

Tables XXIV and XXV display some allocations of the buffer places *inBB*, *inSB*, *inFBB*, and *inFSB*. The first big box arrives in the preassembly buffer after 2 seconds, directly being passed into the working station. As the small boxes only need half the processing time, i. e., 9 seconds, the first small box arrives after 11 seconds. Thus, both boxes are ready at the same time for the final assembly. This can be seen as the buffers for *finish* receive the first boxes after 20 seconds. Because of the final assembly taking 25 seconds, these buffers need to hold the next preassembled boxes ready after 45 seconds. Thus, folding for the second big and small boxes should start at 27 and 36 seconds, respectively.

TABLE XXIV. ALLOCATIONS OF TOKENS ON THE PLACES *inBB* AND *inSB* IN THE EVENT TRIGGERED PULL MODEL AS DEPICTED IN FIGURE 10 AROUND THE FIRST AND LAST SIX SYSTEM STATES WITH TOKEN CHANGES

	inBB				inSB			
state	type	elapsed	count	batchSize	type	elapsed	count	batchSize
1	BB	00:00	0	0	SB	00:00	0	0
2	BB	00:02	1	1	SB	00:11	1	1
3	BB	00:02	0	1	SB	00:11	0	1
4	BB	00:02	0	1	SB	00:11	0	1
5	BB	00:27	1	1	SB	00:36	1	1
6	BB	00:27	0	1	SB	00:36	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
222	BB	30:27	0	1	SB	30:36	0	1
223	BB	30:27	0	1	SB	30:36	0	1
224	BB	30:52	1	1	SB	31:01	1	1
225	BB	30:52	0	1	SB	31:01	0	1
226	BB	30:52	0	1	SB	31:01	0	1
227	BB	30:52	0	1	SB	31:01	0	1

TABLE XXV. ALLOCATIONS OF TOKENS ON THE PLACES *inFBB* AND *inFSB* IN THE EVENT TRIGGERED PULL MODEL AS DEPICTED IN FIGURE 10 AROUND THE FIRST AND LAST SIX SYSTEM STATES WITH TOKEN CHANGES

	inFBB				inFSB			
state	type	elapsed	count	batchSize	type	elapsed	count	batchSize
1	BB	00:00	0	0	SB	00:00	0	0
2	BB	00:00	0	0	SB	00:00	0	0
3	BB	00:20	1	1	SB	00:20	1	1
4	BB	00:20	0	1	SB	00:20	0	1
5	BB	00:20	0	1	SB	00:20	0	1
6	BB	00:45	1	1	SB	00:45	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
222	BB	30:45	1	1	SB	30:45	1	1
223	BB	30:45	0	1	SB	30:45	0	1
224	BB	30:45	0	1	SB	30:45	0	1
225	BB	31:10	1	1	SB	31:10	1	1
226	BB	31:10	0	1	SB	31:10	0	1
227	BB	31:10	0	1	SB	31:10	0	1

Due to the *pull* requests, both unfolded boxes are supplied at the right time in the preassembly buffers. The remainders of the tables shows the allocations for the last system states.

Table XXVI examines the allocations for the last two places, the test buffer *inT* and the finished *goods* store.

TABLE XXVI. ALLOCATIONS OF TOKENS ON THE PLACES *inT* AND *goods* IN THE EVENT TRIGGERED PULL MODEL AS DEPICTED IN FIGURE 10 AROUND THE FIRST AND LAST FIVE SYSTEM STATES WITH TOKEN CHANGES

	inT				goods			
state	type	elapsed	count	batchSize	type	elapsed	count	batchSize
3	FB	00:00	0	0	FB	00:00	0	0
4	FB	00:45	1	1	FB	00:00	0	0
5	FB	00:45	0	1	FB	00:52	1	1
6	FB	00:45	0	1	FB	00:52	1	1
7	FB	01:10	1	1	FB	00:52	1	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
223	FB	31:10	1	1	FB	30:52	73	1
224	FB	31:10	0	1	FB	31:17	74	1
225	FB	31:10	0	1	FB	31:17	74	1
226	FB	31:35	1	1	FB	31:17	74	1
227	FB	31:35	0	1	FB	31:42	75	1

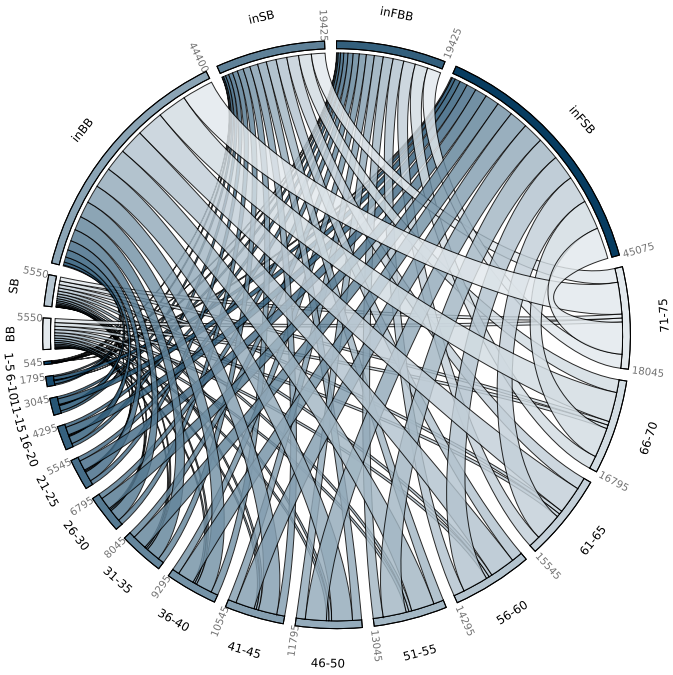

Fig. 11. Accumulated waiting times [in seconds]
in the event triggered push model:
(upper half circle) named storage places
(lower half circle) clusters of five consecutive boxes
(circular area) distribution of accumulated waiting times
of five-box-clusters to individual storage places.

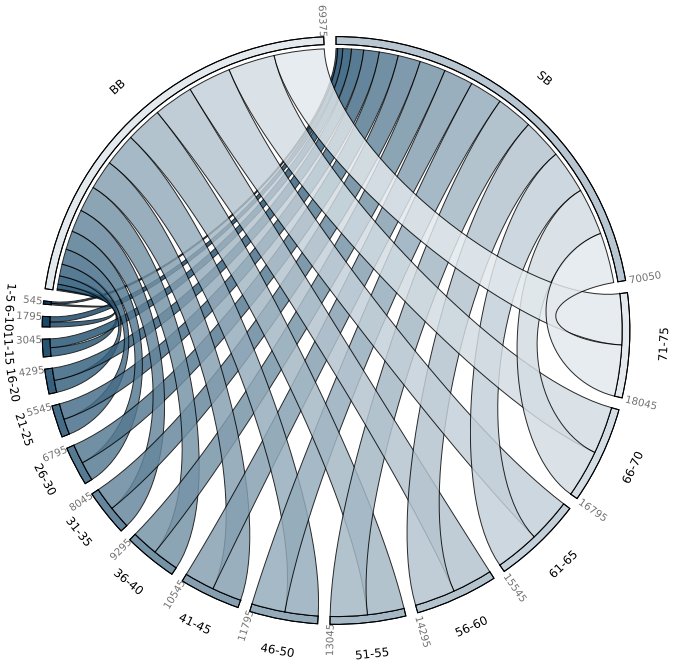

Fig. 12. Accumulated waiting times [in seconds]
in the event triggered pull model:
(upper half circle) named storage places
(lower half circle) clusters of five consecutive boxes
(circular area) distribution of accumulated waiting times
of five-box-clusters to individual storage places.

The first finished box arrives at *inT* after 45 seconds. The quality test starts immediately and the checked box is passed to the outgoing store after a total elapsed time of 52 seconds.

As is the case for the event triggered push model, storage costs are not directly visible in this result set and have to be processed separately. However, the tokens on the finished *goods* warehouses from both models resemble each other.

C. Simulation Results

As expected, the simulation of the two models show no difference in total processing or idle times for single workplaces. Although, differences on storage places become evident again.

Because the distribution of waiting times behind *finish* is the same for both models, only the first five storage places need to be analyzed. Figure 11 shows the allocation of the boxes' waiting times to single stocks - and hence the distribution of storage costs - in the push model while Figure 12 shows the same for the pull model.

The stocks on *material* are split by type (*BB* and *SB* in the figures). The boxes themselves are aggregated into five-box-clusters for clarity. The trend to successively longer waiting times is clearly visible.

The push model's *material* gets cleared as fast as possible. This, however, leads to large buffers (*inBB*, *inSB*) just before the first concurrent production steps. As the preassembly requires less time than the final assembly, the two incoming buffers for *finish* (*inFBB*, *inFSB*) are also highly occupied.

As anticipated, the pull principle leads to zero interim buffer inventory throughout the stocks under consideration. This is clearly visible as, although the total accumulated waiting times for both models are equivalent, the pull model unites them on the main storage *material* only.

This opens up the possibility to externalize costs. One such alternative is to use consignment stores: suppliers maintain warehouses inside the customer's facility and, thus, must keep the material in their balance sheet. Another common possibility is just-in-time delivery. With high process stability and fast response times, even the use of a kanban cycle with the supplier is conceivable [6]. These options may lead to partial or total eradication of stocks.

The final remark of the former section regarding muda and friction between local vs. global optima still holds true.

VIII. Conclusion and Future Work

The advantage of simulating the *Box Game* using clock pulse models is that the fluctuation of stocks can be observed in a very illustrative way while the simulation is running. The amount of items in each storage is clearly visible for every second. Since in the *P-S.C* the color of and the symbol on places can change depending on the number of tokens on the place, the advantages and disadvantages of push and pull can be demonstrated very figuratively. As a consequence, the simulation is a demonstrative extension of the personal experience for the students in the logistics laboratory.

Event triggered models on the other hand emphasize the end results over the visualization during runtime. As such, they show advantages in computation times.

A. Computational Times

Using either of the clock pulse models, the presented scenario runs for 1902 discrete time steps that represent seconds in the *Box Game*. This equals to almost 32 minutes. Although this is a much longer period than one that can be played by students - because concentration levels decrease after about 5 minutes - it seems unnecessary as steps are computed at which no change of the system's state occurs.

Even though, calculating these steps takes time. In Chrome on an iMac (4 GHz Quad-Core Intel Core i7, 16 GB RAM) the full simulation takes on average 8234 milliseconds. The duration for simulating a working day increases linearly.

However, this effort can be reduced drastically if the concrete result is more important than the runtime visualization. In this case, it would be sufficient to calculate new states only in the moments of state changes. Using event triggered models, the simulation for the given scenario can be reduced to 79 steps for the push and 228 steps for the pull version. The difference is due to the kanban calculations needed during runtime. On the aforementioned computer, the simulations only run for 315 and 923 milliseconds on average, respectively. The advantages of event triggered simulation increase more if the demand interval in which changes occur vary strongly from one part of the model to the other since it takes into account local state changes. Hence, the latter approach scales better, making it beneficial for industrial applications.

B. Result Presentation

The primary presentation advantage of the clock pulse models - visualization of state changes during runtime - cannot inherently be conveyed without animation.

Generally, the simulation results of Petri net models can be deducted from the nets' reachability sets. The *P-S.C* provides functionalities for completely or partially logging the actually reached system states, i. e., the simulation results, and for exporting this information in form of comma separated values. For the diagrams presented in Sections VI and VII, such data was used to generate the visualizations in external programs in an individual working step.

Incidentally, this lead to the question as to how to present data to the target audience. However, this audience and their information needs have to be determined in the first place. Modeling and domain expertise often exist disparately, so modelers first need to understand which goals the domain experts want to achieve and which indicators can be used for reaching them.

The cumbersome external processing step and the requirement for an audience orientation combined show the necessity for an internal visualization solution. The goal is to supply modelers with fitting tools in form of an adaptable, integrated dashboard. As a first step, a corresponding research-agenda has been established [36].

C. Key Takeaways

The challenges that had to be overcome for implementing the presented models led to some new insights in the development of conceptual models for discrete time-dependent systems. Eventually, the authors found their personal best practice that consists of the following steps:

1. Define data types for the different stocks and other data objects. Then, initialize the corresponding places in accordance with the starting condition.
2. Augment the model by transitions for beginning and ending specific tasks like delivering raw materials, building or quality testing.
3. Identify the next item to be taken and the moment this will occur. This also allows for implementation of different prioritization strategies.
4. Start with modeling the simpler push principle and augment this model by pull principles.
5. Look for a proper visualization of the simulation results.

Moreover, at this time the development of a clock pulse model may be seen as a preliminary for the development of an event triggered model as some perceive the implementation of the event triggered models as more difficult in comparison. If an event triggered model is needed, the following steps - especially the second one - might be helpful:

1. Develop the clock pulse model first.
2. Observe the reasons for state changes with the aid of the clock pulse model and derive the event triggered model from these observations.
3. Look for a proper visualization of the simulation results.

From personal observations, the authors derive the following suggestion on when to use which modeling approach:

Use a clock pulse simulation if either a clock pulse visualization of the system's states is needed or if the computer is fast enough for the few simulations that must be run for the modeled system.

Use an event triggered simulation if high simulation speed is necessary due the complexity of the modeled system, if fast answers are needed in production, or if a large number of variations of the production schedule or input data has to be compared. In general, the more often a specific model needs to be simulated, the more it is worthwhile to develop an event triggered model instead of a clock pulse model.

In order to ease the step from a clock pulse model to an event triggered model, in the future the *P-S.C* will receive an extension such that users are supported in finding the relevant moments of state changes.

D. What else?

The biggest impression on the authors was the realization that what can be modeled and simulated with the aid of Petri nets is only restricted by the modelers imagination and the ability of the used tool. In opposite to other out of the box modeling environments for logistics, a user is free to lay the focus on any parameter they are interested in most.

However, two further challenges exist: Modelers must be able to develop sophisticated, abstract models and they must find a way to visualize the results. The authors hope to have given a possible answer to the first challenge and see major future work concerning the second one.

The *P-S.C* itself and the lessons learned regarding modeling are used for theoretical and practical research.

The bullwhip effect, for example, is a phenomenon in logistics that is widely known, but rarely examined from a simulation perspective. However, using the *P-S.C* such a simulation becomes manageable, giving insight into the basic mechanism and possibilities to prevent the effect [37].

On the practical side, the authors work on simulating and evaluating the processes used in a large hazardous goods warehouse that is currently be built. The materials in question will be transferred from an older warehouse, however, the established processes can not be adopted due to scale, handling and legal aspects. Thus, it is necessary to examine the system's behavior for problems and anomalies.

Teaching aspects, as shown in the *Box Game*, theoretical and practical research cross-fertilize each other. In effect, this leads to better tools, improved modeling competencies, and more wide-spread usage of simulation and its benefits.

References

[1] S. Haag, L. Zakfeld, C. Simon, and C. Reuter, "Event Triggered Simulation of Push and Pull Processes," in *SIMUL 2020: The Twelfth International Conference on Advances in System Simulation*, L. Parra, Ed., Porto (Portugal), 2020, pp. 68–73.

[2] C. Simon, S. Haag, and L. Zakfeld, "Clock Pulse Modeling and Simulation of Push and Pull Processes in Logistics," in *SIMMaApp: Special Track at SIMUL 2020: The Twelfth International Conference on Advances in System Simulation*, F. Herrmann, Ed., Porto (Portugal), 2020, pp. 31–36.

[3] M. Zhou and K. Venkatesh, *Modeling, Simulation, and Control of Flexible Manufacturing Systems - A Petri net Approach*, ser. Intelligent Control and Intelligent Automation. Singapore, New Jersey: World Scientific, 1999, vol. 6.

[4] U. Dombrowski and T. Mielke, *Ganzheitliche Produktionssysteme: Aktueller Stand und zukünftige Entwicklungen*. München: Springer Vieweg, 2014, German, transl. *Holistic production systems: Current status and future developments.*

[5] J. Gottmann, *Produktionscontrolling: Wertströme und Kosten optimieren*, 2nd ed. Wiesbaden: Springer Gabler, 2019, German, transl. *Production controlling: Optimizing value flows and costs.*

[6] H. Wildemann, *Kanban-Produktionssteuerung*, 28th ed. München: TCW, 2020, German, transl. *Kanban production control.*

[7] L. Recalde, M. Silva, J. Ezpeleta, and E. Teruel, *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. Berlin: Springer, 2004, ch. Petri nets and manufacturing systems: An examples-driven tour, pp. 742–788.

[8] W. Reisig, *Understanding Petri Nets*. Berlin: Springer, 2013.

[9] H. J. Genrich and K. Lautenbach, "System Modelling with High-Level Petri Nets," *Theoretical Computer Science*, vol. 13, 1981.

[10] K. Jensen, *Coloured Petri-Nets*, 1st ed. Berlin: Springer, 1992.

[11] M. Montali and A. Rivkin, "From DB-nets to Coloured Petri Nets with Priorities (Extended Version)," *CoRR*, vol. abs/1904.00058, 2019.

[12] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets," MIT, Project MAC, Technical Report 120, 1974.

[13] P. Merlin, "The Time-Petri-Net and the Recoverability of Processes," University California, Irvine, Tech. Rep., 1974.

[14] J. Sifakis, "Use of petri nets for performance evalutation," in *Measuring, modelling and evalutating computer systems*, ser. IFIP, H. Beilner and E. Gelenbe, Eds., North Holland Publ. Co., 1977, pp. 75–93.

[15] H.-M. Hanisch, *Petri-Netze in der Verfahrenstechnik*. München: Oldenbourg, 1992, German, transl. *Petri nets in process engineering.*

[16] H.-M. Hanisch, "Dynamik von Koordinierungssteuerungen in diskontinuierlichen verfahrenstechnischen Systemen," in *Petrinetze in der Automatisierungstechnik*, E. Schnieder, Ed. München, Wien: Oldenbourg Verlag, 1992, German, transl. *Dynamics of coordination controls in discontinuous process engineering systems.*

[17] R. König and L. Quäck, *Petri-Netze in der Steuerungs- und Digitaltechnik*. München, Wien: Oldenbourg Verlag, 1988, German, transl. *Petri nets in control and digital technology.*

[18] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzè, "A unified high-level petri net formalism for time-critical systems," *IEEE Transactions On Software Engineering*, vol. 17, no. 2, pp. 160–172, 1991.

[19] H.-M. Hanisch, K. Lautenbach, C. Simon, and J. Thieme, "Timestamp Nets in Technical Applications," in *IEEE International Workshop on Discrete Event Systems*, San Diego, CA, 1998.

[20] C. Simon, "Developing Software Controllers with Petri Nets and a Logic of Actions," in *IEEE International Conference on Robotics and Automation, ICRA 2001*, Seoul, Korea, 2001.

[21] K. Jensen, "High-Level Petri Nets," *Informatik-Fachberichte*, vol. 66, pp. 166–180, 1983.

[22] L. Popova-Zeugmann, *Time and Petri Nets*. Berlin: Springer, 2013.

[23] C. E. Knoeppel, *Installing Efficiency Methods*. The Engineering Magazine, 1915.

[24] T. Ohno, *Toyota Production System*. Milton Park, UK: Taylor & Francis, 1988.

[25] BPMI, "BPMN 1.0 - Business Process Model and Notation," https://www.omg.org/spec/BPMN/ (last accessed 2021.11.20), 2004.

[26] OMG, "BPMN 2.0 - Business Process Model and Notation," http://www.bpmn.org/ (last accessed 2021.11.20), 2011.

[27] C. Simon, "Web-Based Simulation Of Production Schedules With High-Level Petri Nets," in *32rd International ECMS Conference on Modelling and Simulation (ECMS 2018)*, L. Nolle, A. Burger, C. Tholen, J. Werner, and J. Wellhausen, Eds. Wilhelmshaven, Germany: SCS Europe, 2018, pp. 275–281.

[28] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design Science in Information Systems Research," *MIS Q.*, vol. 28, no. 1, pp. 75–105, mar 2004.

[29] C. Simon and S. Haag, "Simulatable Reference Models To Transform Enterprises For The Digital Age – A Case Study," in *ECMS 2020: 34th International ECMS Conference on Modelling and Simulation*, M. Steglich, C. Müller, G. Neumann, and M. Walther, Eds., 2020, pp. 294 – 300.

[30] C. Simon and S. Haag, "A Case-Study to Teach Process-Aware Information Systems," *EMISA Forum: Proceedings of the SIG Enterprise Modelling and Information Systems Architectures of the German Informatics Society*, vol. 40, pp. 9–10, 2020.

[31] S. Haag and C. Simon, "Simulation of Horizontal and Vertical Integration in Digital Twins," in *ECMS 2019: 33rd International ECMS Conference on Modelling and Simulation*, M. Iacono, F. Palmieri, M. Gribaudo, and M. Ficco, Eds., 2019, pp. 284 – 289.

[32] C. Simon and S. Haag, "Simulation vertikaler Integration: Vom Top-Floor zum Shop-Floor und zurück," in *Tagungsband AKWI*, T. Barton, F. Herrmann, V. G. Meister, C. Müller, C. Seel, and U. Steffens, Eds., 2018, pp. 104 – 113, German, transl. *Simulation of vertical integration: from top floor to shop floor and back again.*

[33] C. Simon, "Eine Petri-Netz-Programmiersprache und Anwendungen in der Produktion," in *Tagungsband AKWI*, T. Barton, F. Herrmann, V. G. Meister, C. Müller, and C. Seel, Eds., 2017, pp. 61–70, a Petri net programming language and applications in production.

[34] C. Simon, S. Haag, and L. Zakfeld, "Simulation taktgesteuerter Modelle von Push- und Pull-Prozessen in der Logistik," *Anwendungen und Konzepte der Wirtschaftsinformatik*, vol. 13, pp. 27–33, 2021, German, transl. *Simulation of clock-controlled models of push and pull processes in logistics.*

[35] Petri Nets World, "Petri Nets Tools Database Quick Overview," https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html (last accessed 2021.11.20), 2020.

[36] C. Simon, S. Haag, and L. Zakfeld, "Research-Agenda for Process Simulation Dashboards," in *ECMS 2021: 35th International ECMS Conference on Modelling and Simulation*, 2021, pp. 243–249.

[37] C. Simon, L. Zakfeld, C. E. Jensen, D. Klietsch, and M. Montag, "Can simulation prevent companies from the bullwhip trap? New approaches to model the bullwhip effect with the aid of Excel and high-level Petri nets," in *SIM-SC : Special Tack at SIMUL 2021 : The Thirteenth International Conference on Advances in System Simulation*, F. Herrmann, M. Popescu, and M. Audette, Eds., 2021, pp. 31–37.