Simulation in Production
and Logistics 2015
Markus Rabe & Uwe Clausen (eds.)
Fraunhofer IRB Verlag, Stuttgart 2015

Discrete Event Simulation of Modular Production System Models using Petri Nets

Ereignisdiskrete Simulation modularer Produktionssystemmodelle mittels Petri-Netzen

Markus Rabe, Maik Deininger, TU Dortmund, Dortmund (Germany),
markus.rabe@tu-dortmund.de, maik.deininger@tu-dortmund.de

Abstract: Medium-term production planning is a complex task for manufacturers. The question whether accepting or declining a new order often cannot be answered well-grounded. Additionally, it cannot be predicted, which consequences an acceptance will have on the running production. This paper aims to provide a basis for medium-term production planning using Petri nets and discrete event simulation. Petri nets are used for creating Modular Production System Models (MPSM). MPSM are utilised for representing any process that exists within the considered production system as also processes that may be implemented to adapt to new requests. As consequence, the existing production system is modelled by interconnecting MPSMs and is varied by adding or removing modules. Each model is analysed using discrete event simulation based on Petri nets. Thus it can be verified whether all orders can be fulfilled or not.

1 Introduction

Production planning is a challenging task, especially for manufactures with custom-made products. The effects of accepting or declining an order without knowing how the production system is affected often cannot be forecasted adequately. In order to support planners with this decision, a method needs to be developed that is able to change and evaluate production systems automatically and provide the planner with the information needed. The changes made have to be target-oriented to avoid evaluating systems that do not reduce the target parameters as make span or costs. Based on such a method, a tool can be implemented to support the decision making within production planning.

Simulation is a wide-spread technique for evaluating existing or planned production systems as well as for analysing the impact of alterations to such systems (cp. Lefrançois et al. 1996; Kara et al. 2007; Frank et al. 2013). Since state changes in these systems in general occur at discrete points in time, Discrete Event Simulation (DES) is used in most cases. Different tools exist on the market that enable planners

to model and simulate production systems. The simulation models are usually created by connecting various building blocks manually. Some of the tools support automatic changes, but these changes are not operated through a superior mechanism as for example a heuristic optimisation that evaluates and selects the changes made. Such an algorithm needs a basis to work with. Therefore, a modular approach is investigated where a production system is interpreted as a compilation of modules. A first attempt was made by Rabe and Deininger (2013). This paper builds up on this work and uses Petri nets (PNs) for creating Modular Production System Models (MPSMs), where each module handles a single order. By connecting single modules, the path a customer order takes through the production system can be represented. Furthermore, by adding, exchanging, or removing modules, the system can be changed easily.

This paper is structured as follows: the next section gives a brief introduction to modelling and discrete event simulation. Afterwards the PN type used in this paper is introduced, before creating and simulating MPSMs in section 4 with PNs. The paper is closed with a conclusion that summarises the results and gives a short outlook.

2 Modelling and Simulation of Production Systems

In production and logistics simulation is one of the most important techniques. It is used for analysing and optimising existing as well as planned production systems. Due to the nature of these systems, where changes of interest only occur at discrete points in time, the most applied simulation method is DES as can be seen in many publications (cp. Lefrançois et al. 1996; März and Krug 2011). Tools like Plant Simulation (cp. Bangsow 2010) provide building blocks that are used to create complex simulation models. In general, the material flow is modelled and each building block represents an individual segment. These segments are means of production, means of transport, or resources. Additionally, if the provided building block does not meet the requirements, it is possible to modify existing or create new elements using a domain-specific language as for example SimTalk in Plant Simulation. Once the model is created, a DES can be used for analyses. In this paper, Petri Nets (PNs) are utilised to create modules that can be connected to model larger systems. These modules represent the flow of customer orders and internal orders instead of the material flow.

PNs are used to model various systems (cp. Viswanadham and Raghavan 2000; Dahms and Schmidt 2005; Koch 2014). A PN is a directed graph $PN = (T, P, E)$ where T is a disjoint finite set of transitions, P is a disjoint finite set of places, and $E \subseteq (P \times T) \cup (T \times P)$ is a finite set of directed arcs where each connects a transition with a place (Priese and Wimmel 2008; Reisig 2010). The concept of object orientation was introduced to PNs based on the concepts used in programming (cp. Desel, J. et al. 2004). There are different approaches for using object orientation in PNs (cp. Ceska et al. 1998; Dahms and Schmidt 2005; Koci et al. 2008). The most common is to treat tokens as instances of classes having member variables and functions. Using this approach, tokens contain the data that are manipulated through transitions they passed. Another approach is introduced with Object Oriented Petri Nets (OOPN), where PNs are organized in classes consisting of an object net and a set of method nets (Koci et al. 2008). OOPNs support

inheritance. Subclasses may redefine or add transitions and places as well as new methods. A token can either be a trivial object, e.g. a number, or an instance of an OOPN. A more complex type with higher flexibility is defined by Timed Hierarchical Object-Related Nets (THORNs) (Schöf et al. 1997) that are introduced within section 3 in detail.

By using THORNs the modeller is able to adapt predefined modules at two abstraction levels. It is possible to change the behaviour of a module by adding or removing places and transitions. Additionally, since THORNs use C++ for defining the behaviour of each element, individual elements can be implemented and used.

The simulation of PN models can be conducted using DES (cp. Zhou and Jeng 1998). Basically, there are two events: the transition fire event that consumes and produces tokens at places and the token move event that activates or deactivates transitions. How this concept is applied to THORNs is shown in section 4.

3 A Special Petri Net Type: Timed Hierarchical Object-Related Net

The PN type used in this paper is a THORN (cp. Wieting 1996; Schöf et al. 1997). THORNs extend PN by adding special types of places, transitions, and arcs, whereas places also offer a structure, as depicted in Figure 1. Additionally, object orientation is introduced not only to tokens but also for places, transitions, and arcs. Thus, a THORN can be used within many fields of application. THORNs implement two time concepts: duration of firing and a variation of window of invalidity (Starke 1995). Both times are determined using functions and thus can be computed dynamically.

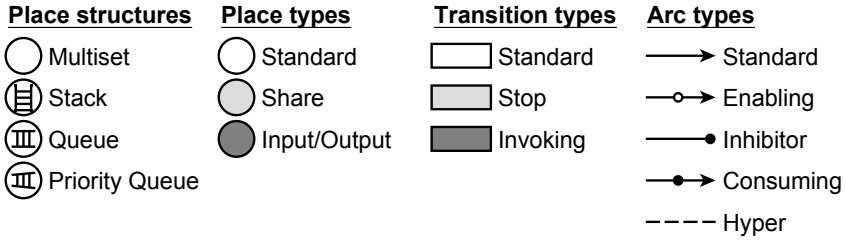

Figure 1: *THORN elements*

A transition contains multiple mandatory functions. Most important for the behaveiour of the THORN is the condition function. In order to activate the transition, the pre and post area conditions, as in standard PNs, need to be fulfilled and the condition function needs to be evaluated as true. This function enables the modeller to create transitions that are only activated if the tokens in the pre area meet specific conditions. For example, a token representing a worker can be tested for its qualification and working time spend within the current shift.

The three transition types indicate which kind of action is applied. Standard transitions manipulate or create new tokens, invoking transitions create new subnets that

exist and run parallel to the main THORN, and stop transitions destroy the subnet they are part of. The main THORN does not contain stop transitions.

Each transition always matches one of four states (inactive, delayed, completely delayed or active) and is only allowed to fire while being active. A transition is inactive if the pre area condition is not fulfilled. It is delayed if the pre area condition is fulfilled and completely delayed if it was delayed for a specified time (window of invalidity). If a transition is completely delayed and its post area condition is fulfilled, it switches to the state active where it is allowed to fire.

Next to the condition function of a transition, the place structure has a huge impact on the behaviour of the net. While multiset places allow taking any suitable token, stack, queue, and priority queue places only allow using the first token in line. A stack place implements the last in first out concept and may be used for modelling loading and unloading of means of transport. Queue and priority queue places apply first in first out, whereas priority queue places additionally allow sorting the tokens through a priority function. This can be used for defining in which sequence orders shall be processed.

The type of a place indicates where the tokens are used. Tokens from share places are equally used in multiple, parallel existing THORNs. Input and output places are used to transfer tokens between THORNs when creating or destroying a subnet. Tokens on standard places are only used within a single THORN.

Arcs influence the fulfilment of the pre and post area conditions. Five types are available. A standard arc can be used as input arc (place to transition) and as output arc (transition to place). It also can be weighted and thus determine how many tokens have to be moved. The enabling arc is always an input arc and can have a weight. In contrast to the standard arc, it only is tested whether the right amount of tokens exist on the place or not. They are not moved with the firing of the transition. Inhibitor and consuming arcs have no weight attached and only can be used as input arcs. The inhibitor arc inhibits the activation of a transition in case the connected place contains any token. The consuming arc activates a transition if the place contains at least one token. All tokens are removed from the place, the moment the connected transition fires. A hyper arc is used to connect share places with invoking transitions and thus indicating that the place is used within the subnets being created. Hyper arcs are undirected and in general are interpreted as a combined input and output arc. Within the subnet the share place is connected to transitions using arcs of the other four types.

4 Simulating Petri Nets without Conditional Events

The simulation of THORNs is based on a DES where the events are connected to places and transitions. The events are:

- a token arrives at a place (EAdd),
- a token is removed from a place (ERemove),
- a transition changes its state (EStateChange),
- a transition is completely delayed (EDelay), and
- a transition fires (EFire)

where EAdd, ERemove, EDelay, and EFire are booked events and EStateChange is a conditional event (for booked and conditional events see e.g., Robinson 2004). For example, a transition fires at $t = t_0$. Thus, the following events are created: at time t_0 ERemove for each place within the pre area and at time $t = t_0 + t_{Fire}$ EAdd for each place within the post area. Since only places connected to the transition are affected, it is not necessary to validate each EStateChange but only those that are connected to the places affected. If a transition's state changes to delayed, then an EDelay event is created with execution time $t = t_0 + t_{Delay}$. As a consequence, EStateChange can be integrated with EAdd and ERemove and is not needed as a separate event. EAdd now additionally validates the states of transitions in the post area and ERemove validates the states of transitions in the pre area of the place.

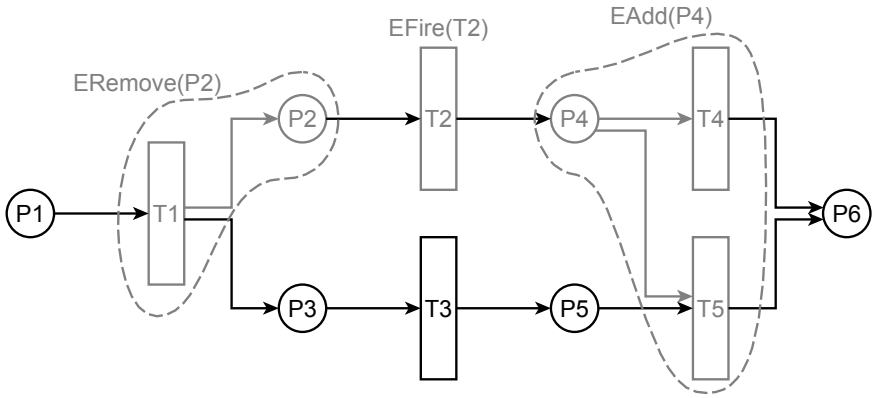

Figure 2: *Transitions tested within ERemove and EAdd event*

Figure 2 illustrates which transitions and places are handled by the events ERemove and EAdd when transition T2 fires. The remove event of place P2 includes a status update of transition T1 and the arrive event of place P4 includes the status updates of transitions T4 and T5. Thus, THORNs can be simulated by using four types of events. Through creating the initial mark-up using EAdd events, instead of just "placing" tokens, and ERemove events, for places connected by inhibitor arcs to transitions, at $t = t_0$, the simulation can be started without using conditional events.

Table 1: *Transitions to be checked if changes apply to its pre area places*

	Token added to place				Token removed from place			
	Multi-set	Stack	Queue	Priority queue	Multi-set	Stack	Queue	Priority queue
Inactive	+	+	+	+	–	+	+	+
Delayed	–	+	–	+	+	+	+	+
Completely delayed	–	+	–	+	+	+	+	+
Active	–	+	–	+	+	+	+	+

Furthermore, based on the type of a place and the state of the transition, it can be decided, if the transition is affected and may be excluded from state validation. Table 1 and Table 2 show whether a transition needs to be checked (+) or not (–). Considering that in THORNs each transition has its own additional condition function, which can be very complex when checking and comparing specific token attributes, avoiding this test can improve the simulation speed, significantly.

For example, adding a token to a stack place of the pre area of a transition, the transition has always to be evaluated. Since the new token is available for use instead of the old one, the conditional function may lead to a different result. Adding a token to a multiset place in the pre area, inactive transitions have to be evaluated only. All tokens are still available for use and e.g. delayed transitions cannot become inactive.

Table 2: *Transitions to be checked if changes apply to its post area places*

	Token added to place				Token removed from place			
	Multi-set	Stack	Queue	Priority queue	Multi-set	Stack	Queue	Priority queue
Inactive	–	–	–	–	–	–	–	–
Delayed	–	–	–	–	–	–	–	–
Completely delayed	–	–	–	–	+	+	+	+
Active	+	+	+	+	–	–	–	–

As Table 2 indicates, only few changes within the post area demand an evaluation of the transition's state. This is due to the fact that only the capacity and number of tokens at the place influence the transitions state. Adding a token may lead insufficient free capacity. Thus, only active transitions may change back to the state of being completely delayed. As the consequence, a completely delayed transition may be activated by removing a token. The states inactive and delayed do not take the post area into account and, therefore, are not affected by changes.

5 Modular Production System Models with Petri Nets

For modelling a production system, the system is considered as an aggregation of individual modules. Each module performs a single task and competes with other modules for resources such as workers, means of production, and materials. Figure 3a depicts the simplest form of a module. It takes an order (O1) and uses workers (W), means of production (P), and material (M). As output, a new order (O2) is created. Additionally, used resources are put back to their places and produced resources are placed at the appropriate places. For a more detailed specification of the performed actions, an invoking transition can be used that allows for creating a new subnet and now uses the places W, P, and M as share places instead of standard places. The subnet has its own places and transitions that specify how the order is handled. The changed module is shown in Figure 3b. Also a refinement of the transition could be used, but the invoking transition provides some advantages:

- multiple instances of the subnet are allowed (capacity > 1),
- with a capacity of one, the transition cannot be activated before the order is completely handled, and
- the subnet can be changed without touching the main net.

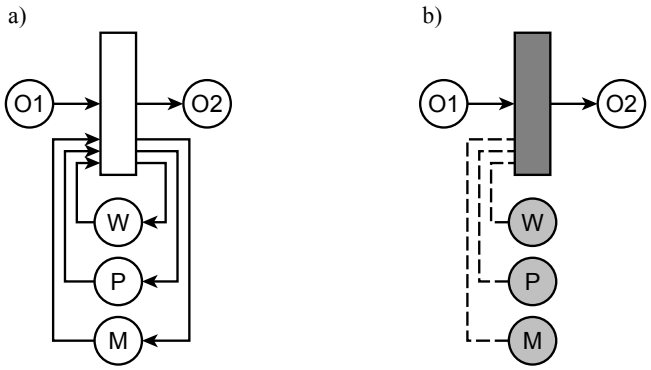

Figure 3: *Simple module (a) and simple module using an invoking transition (b)*

The modeller has to create a model, called MPSM, for each order that can be handled by the production system. These models are combined to a model of the complete production system by connecting places that store the same type of tokens.

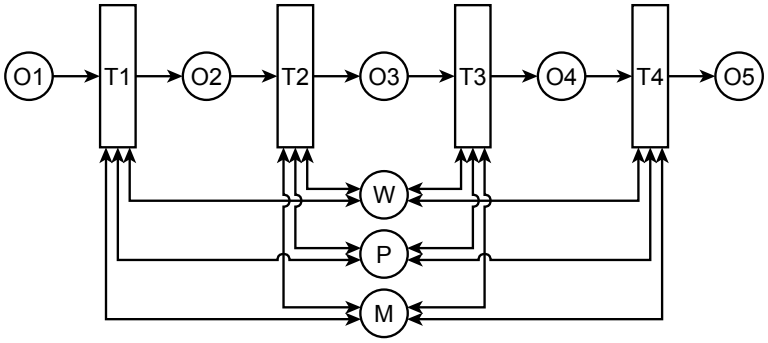

Figure 4: *Combination of modules*

For illustration, a production line that consists of four machines where each represents a single step of the production is used. Here, each machine can be modelled as a module as depicted in Figure 3a. The first module receives an order for processing step one and creates an order that activates module two which processes it. Each of the modules uses the same places for workers, means of production, and material and, therefore, they are competing for the resources. Figure 4 depicts this production line, where the double-headed arrow indicates the combination of a standard input and output arc. Invoking transitions can be used for refine-

ment and thus to improve the accuracy and to create different views: e.g. one for the manager and another one for the worker.

If analyses show that customer orders cannot be handled properly, the modular approach allows for easily adding or exchanging modules. Assuming the bottleneck is in handling the orders at place O2, a new transition T5 can be added, as shown in Figure 5. Here, transition T5 represents a production step that uses a new machine that is capable of performing the same steps as the two machines used in transitions T2 and T3. The places for resources (W, P, and M) are omitted for clarity.

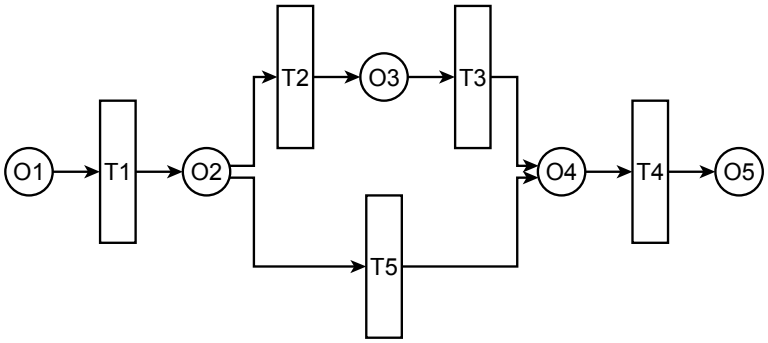

Figure 5: *Modified production line*

In reality, the acquisition of a machine is not only adding it to the production system. Among other things, the new machine, as well as the workers using it, needs to be installed and, if necessary, qualified. Therefore, the token representing the machine is not available for production since it has not the required qualifications. This has to be modelled by an individual module which exists in parallel to the model of the production system. This qualification module will use resources that initially had been scheduled for the production and are now no longer available. This conflict has to be solved. One opportunity is creating an additional production order for materials produced by machine 1 (represented by transition T1). As consequence, the work load for machine 1 is enlarged and the production has to be started earlier. Another option is adding a new module that is capable of performing the same steps as machine 1.

For evaluating different scenarios, a heuristic optimisation shall be developed that varies the model and analyses each version through simulation automatically. If shortages occur, those will be solved by adding MPSMs. Therefore, the simulation will be stopped. By predefined runtimes for each of the modules, it will be gone back in simulation, using backups that were made during the simulation run. Thus, a suitable point in time where the module has to be added can be selected. Now, the simulation can be restarted and the new module will support the production and the conflict will not arise.

6 Conclusion

This paper introduces an approach for applying discrete event simulation of THORNs without using conditional events. Therefore, the events that depend on conditions, as state changes of transitions, are integrated with booked events. The events of adding or removing tokens trigger the evaluation of a transition's state directly. Thus, it can be avoided to test each transition in every single time step and improve the performance of the simulation.

Furthermore, this simulation approach is used to evaluate models created with MPSMs. MPSMs are small models described through THORNs representing the handling of orders. By connecting MPSMs, the flow of orders within a production system can be modelled and afterwards analysed using DES. Single modules can easily be added, removed, or exchanged, which allows for evaluating multiple alternative systems.

Further research has to be conducted for developing a heuristic optimisation that enables an automated variation and evaluation of production systems. For this purpose, the MPSMs have to be stored in an appropriate manner: MPSM based models have to be adaptable within the simulation run and the simulation needs to be able to be stopped and restarted at suitable points in time. If these problems can be solved, a production planner will be able to decide if and how all customer orders can be handled.

References

Bangsow, S.: Manufacturing simulation with Plant Simulation and Simtalk. Berlin, London: Springer 2010.

Ceska, M.; Janousek, V.; Vojnar, T.: Object-oriented Petri nets, their simulation, and analysis. In: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics. Piscataway, NJ, USA: IEEE 1998, pp. 262–267.

Dahms, M.; Schmidt, M.: Modeling of dispatching-rules for Job Shop Scheduling in manufacturing systems – A Petri net approach. In: Proceedings of the IEEE International Conference on Systems, Man and Cybernetics. Waikoloa, Hawaii, USA, October 10-12 2005. Piscataway, NJ, USA: IEEE 2005, pp. 2025–2030.

Desel, J.; Reisig, W.; Rozenberg, G.: Lectures on concurrency and Petri nets: Advances in Petri nets. Berlin, New York (NY): Springer 2004.

Frank, M.; Laroque, C.; Uhlig, T.: Reducing computation time in simulations-based optimization of manufacturing systems. In: Pasupathy, R.; Kim, S.-H.; Tolk, A.; Hill, R.; Kuhl, M.E. (Eds.): Proceedings of the 2013 Winter Simulation Conference. Piscataway: IEEE 2013, pp. 2710–2721.

Kara, S.; Rugrungruang, F.; Kaebernick, H.: Simulation modelling of reverse logistics networks. International Journal of Production Economics 106 (2007) 1, pp. 61–69.

Koch, I.: Petrinetze in der Systembiologie. Informatik-Spektrum 37 (2014) 3, pp. 211–219.

Koci, R.; Janousek, V.; Zboril, F., Jr.: Object oriented Petri nets modelling techniques case study. In: Al-Dabass, D.; Nagar, A.; Tawfik, H.; Abraham, A.; Zobel, R. (Eds.): Proceedings of the Second UKSim European Symposium on

Computer Modelling and Simulation. Los Alamitos (CA): IEEE 2008, pp. 165–170.

Lefrançois, P.; Harvey, S.; Montreuil, B.; Moussa, B.: Modelling and simulation of fabrication and assembly plants: An object-driven approach. Journal of Intelligent Manufacturing 7 (1996) 6, pp. 467–478.

März, L.; Krug, W.: Kopplung von Simulation und Optimierung. In: März, L.; Krug, W.; Rose, O.; Weigert, G. (Eds.): Simulation und Optimierung in Produktion und Logistik. Berlin, Heidelberg: Springer 2011, pp. 41–45.

Priese, L.; Wimmel, H.: Petri-Netze. Berlin, Heidelberg: Springer 2008.

Rabe, M.; Deininger, M.: Fabrikmodelle für Job-Shop-Scheduling-Algorithmen in Changing-Steady-State-Systemen. In: Dangelmaier, W.; Laroque, C.; Klaas, A. (eds.): Proceedings of 15. ASIM Fachtagung Simulation in Produktion und Logistik. Paderborn: Heinz-Nixdorf-Inst. 2013, pp. 579–589.

Reisig, W.: Petrinetze: Modellierungstechnik, Analysemethoden, Fallstudien. Wiesbaden: Vieweg + Teubner 2010.

Robinson, S.: Simulation: The practice of model development and use. Chichester, England: Wiley 2004.

Schöf, S.; Wieting, R.; Sonnenschein, M.: Distributed Net Simulation: DNS; Abschlußbericht: OFFIS, Oldenburger Forschungs- und Entwicklungsinstitut für Informatik-Werkzeuge und -Systeme, Forschungsbereich 4, Systemmodellierung 1997.

Starke, P.H.: A memo on time constraints in Petri nets. Berlin, HU Berlin, Institut für Informatik 1995.

Viswanadham, N.; Raghavan, S.N.: Performance analysis and design of supply chains: A Petri net approach. Journal of the Operational Research Society 51 (2000) 10, pp. 1158–1169.

Wieting, R.: Handbuch zur THORN Entwicklungsumgebung. Oldenburg: OFFIS 1996.

Zhou, M.; Jeng, M.D.: Modeling, analysis, simulation, scheduling, and control of semiconductor manufacturing systems: A Petri net approach. IEEE Transactions on Semiconductor Manufacturing 11 (1998) 3, pp. 333–357.