A Petri Net Modeling Framework for the Control of Flexible Manufacturing Systems

Philipp Wenzelburger* Frank Allgöwer*

*Institute for Systems Theory and Automatic Control, University of Stuttgart, Germany.
e-mail: {philipp.wenzelburger, frank.allgower}@ist.uni-stuttgart.de

Abstract: The changes in manufacturing practice towards increasingly flexible structures demand novel strategies for production scheduling and control. To this end, we introduce an automated model generation scheme that builds upon a coarse description of the production process and the factory. The generated model is based on a Petri Net representation of the production process, which we slightly enhance to separate automatically evolving processes from consciously taken manufacturing decisions. From the Petri Net model we deduce a state space description which can be used for the synthesis of feedback control for the manufacturing system. The model generation is illustrated with an example and further possibilities to utilize and enhance the model are discussed. Copyright © 2019 IFAC

Keywords: Flexible manufacturing systems, Industry 4.0, Petri Nets, Production control.

1. INTRODUCTION

Industrial production principles are constantly changing due to competition and the availability of new technologies. At present, the demand for individualized products drives the development towards more flexible manufacturing systems that are capable to quickly adjust to the changing requirements. At the same time, advancements in information and communication technologies open new possibilities to enhance industrial production. Under the name Industry 4.0, research and development on this topic already bears fruit. On the one hand, new paradigms wear down the classical automation pyramid and offer versatile structures providing more flexibility (Bauernhansl, 2014). On the other hand, skill based programming enables abstraction of manufacturing tasks and offers the possibility to address the capabilities of a flexible manufacturing unit on a higher level of abstraction (Thomas et al., 2013; Heim et al., 2015). The manufacturing units’ actions are then executed autonomously with the help of numerous sensors and subordinate logic. However, the linking element in between is still missing. It is not yet known, how the abilities of the robots can be used to implement the new automation paradigms and create truly flexible manufacturing systems. In this paper, we present an intuitive and yet formal approach to link those two sides with the help of Petri Nets. The connection is made by introducing algorithms which automatically generate a Petri Net model from an elementary description of the manufacturing system, whereof a state space description can be deduced which on the one hand allows to represent the flexibility present in the manufacturing setup and on the other hand abstracts the capabilities of the machines on a level compatible with skill based programming. In order to determine the decision variables of the state space model, a specific distinction between two classes of transitions in general Petri Nets is introduced. The state space model will be the basis to apply feedback control and online optimization to leverage the benefits granted by the flexible manufacturing setup and to create a production schedule in terms of skills, which can be executed by the flexible manufacturing units.

In this paper we first introduce the preliminaries of Industry 4.0 and Petri Nets in Section 2, where we also briefly state our notion of flexibility in a manufacturing system. In Section 3, we formulate the manufacturing problem to solve from an application point of view, before we present algorithms which automatically transform it into a Petri Net in Section 4 and one step further into a state space description in Section 5. In Section 6, we discuss the presented results and give an outlook on further improvements and applications.

2. PRELIMINARIES

The focus of Industry 4.0 is on the change of industrial production (Kagermann et al., 2013), to which we aim to contribute with control theoretic methods. We will take the notion of the digital twin as presented in Grieves (2014) as given and assume that this kind of information is always available. This notion implies that for an object in the
physical world there exists a digital representation that holds all the information that accumulate during its lifetime. The digital twin of an object is created with its production plan, expanded with the production parameters used in its fabrication, and is updated during all its lifetime. For a machine it holds for example the skills it can perform, in this paper represented as a list of tasks it can execute, and for an order of a personalized item there is a plan holding the different production steps that have to be fulfilled to manufacture it.

To discuss flexibility of a manufacturing system, it needs to be clarified what „flexibility“ means. In literature many different notions exist (see Beach et al. (2000) and Kim et al. (2003) for an overview), which vary in the level of abstraction, consider different effects in the manufacturing process and can be exploited in different ways. Generally a flexible manufacturing system needs to provide the possibility to change, which can be exploited to adjust to the present requirements in order to increase the efficiency of the production. Those changes are in particular the possibility to

a) execute a task on different machines
b) prioritize different products against each other when they have to be produced on the same machine
c) change the sequence in which the production steps are performed, if this is technically possible.

There is plenty of related work on scheduling in manufacturing and especially in Industry 4.0 scenarios (Dolgui et al., 2019; Liu et al., 2018; Chaudhry and Khan, 2016). Most of it requires detailed information on the production setup, which needs to be considered explicitly in the mathematical formulation and solution of the problem. Hence, when applying those methods, expert knowledge and time for model creation is required. To circumvent this, we consider a simplified approach by abstracting many details on the execution dynamics. This makes it possible to generate the mathematical model automatically from a simplified problem description, which is one of the main intents of our approach. Moreover, we don’t consider the solution of the scheduling problem yet, which is the main focus of many related papers. More general techniques introduced for automatic scheduling in Industry 4.0 (e.g., Rossit et al. (2018)) are applicable to the model presented in this paper and might be considered in future work.

In order to develop and apply control theoretic methods, we need to formulate the problem mathematically, which will be done in the Sections 3 - 5. We choose a model based framework, which on the one hand needs to enable to formulate the flexible aspects of the system, and on the other hand has to have a mathematical formalization. Our choice are Petri Nets, which will be introduced in the sequel and are well suited to describe Industry 4.0 scenario as investigated in Long et al. (2015). Petri Net models are especially suitable for our purposes since they have an algebraic description of the system dynamics which allows to employ existing methods from systems and control theory.

Petri Nets have originally been developed in the 1960s by Carl Adam Petri (Petri, 1962) and have been widely used and enhanced since then. There exists a vast amount of literature that can be consulted, whereof we want to refer to Cassandras and Lafortune (2009) and Seatzu et al. (2013) to get a detailed insight in the topic. For our purposes, a short definition of a very basic form of a Petri Net suffices.

**Definition 1.**
A Petri Net is a tuple $PN = (P, T, E, w, x^0)$, where

$P = \{p_1, \ldots, p_n\}$ is a finite set of places, $n \in \mathbb{N}$, graphically represented as circles,

$T = \{t_1, \ldots, t_m\}$ is a finite set of transitions, $m \in \mathbb{N}$, graphically represented as bars,

$E \subseteq (P \times T) \cup (T \times P)$ is a set of arcs from places to transitions and from transitions to places, graphically represented as arrows,

$w : E \rightarrow \mathbb{N}$ is an arc weight function, graphically represented as numbers labeling the arcs (if an arc has the weight 1 it is not labeled) and

$x^0 \in \mathbb{N}_0^n$ is the initial marking of the Petri Net, from now on called initial state.

The initial state $x^0$ of the place $p_i$ describes the number of tokens in the place $p_i$ at initialization, graphically represented by $x^0$ dots in the corresponding circle. The dynamics of a Petri Net is driven by the so called firing of the transitions and described by the state signal $x : \mathbb{N}_0 \rightarrow \mathbb{N}_0^n$, where $x(k)$ is the state of the Petri Net at time $k$ and $x(0) = x^0$. If a transition $t_j$ fires, tokens are moved between the places according to the weights of the arcs connected to $t_j$. This can be compactly described by introducing the incidence matrix of a Petri Net $B = B^+ - B^-$ composed of the matrices $B^+$ and $B^-$ with the entries $B^+_{ij} = w(t_j, p_i)$ and $B^-_{ij} = w(p_i, t_j)$, and the firing signal $u : \mathbb{N}_0 \rightarrow \mathbb{N}_0^n$. The firing count vector $u(k)$ holds the number of times each transition in the Petri Net fires at time $k$, leading to the Petri Net dynamics

$$x(k+1) = x(k) + Bu(k).$$

Since the places cannot hold a negative number of tokens, the firing of a transition is not always possible. A transition $t_j$ is only enabled, when all places $p_i$, $i = 1, \ldots, n$ hold at least as many tokens as the weight of the arc leading from the place $p_i$ to the transition $t_j$, i.e., $x_i(k) \geq w(p_i, t_j)$ for all $i = 1, \ldots, n$. This can be compactly described through

$$0 \leq x(k) - B^- u(k).$$

Up to this point, we stick to a usual formulation of the Petri Net dynamics as for example used in Cassandras and Lafortune (2009). We treat the firings of the transitions as the means to influence the Petri Net and do not assume any inherent firing rule. The Petri Net dynamics (1) is a linear discrete time system $x(k+1) = Ax(k) + Bu(k)$ with the matrix $A = I$ which is constrained by (2) in order to obey the rule that the state of the Petri Net remains nonnegative.

For the description of our manufacturing problem in form of a Petri Net in Section 3, we will exploit the possibilities of changing the matrix $A$. We introduce an independent part of the Petri Net which holds transitions that fire as soon as they are enabled and will be represented in the matrix $A$ in contrast to the controlled part of the Petri Net which remains the basis for the incidence matrix $B$. Since the independent part of the Petri Net also has to obey the rule that the state vector only holds positive integers,
3. PROBLEM FORMULATION

Having introduced Petri Nets as the framework in which we will model manufacturing systems, we now describe how it can be applied to automatically generate a model of a production system. The class of manufacturing problems we investigate is a generalization of the description of non-cyclic manufacturing systems in Zhou (2012); where, in contrast to our setup, the sequence of tasks is completely fixed. In our abstract view of a production system, the main components are

- A set $\mathcal{T} = \{\tau_1, ..., \tau_n\}$ of tasks which can be executed in the manufacturing system. Each task $\tau$ might depend on other tasks, which need to be finished before $\tau$ can be started.
- A given set $\mathcal{M} = \{m_1, ..., m_{\mathcal{N}}\}$ of production facilities (machines, robots, automated guided vehicles, ...), from now on called machines for brevity. Every machine $m$ can execute a set $\mathcal{T}_m \subset \mathcal{T}$ of tasks.

The machines might differ from one another and the tasks most certainly vary in complexity. Therefore, in addition to the list of task $\mathcal{T}_m$ every machine $m \in \mathcal{M}$ can perform, it is necessary to provide the production times $k_P(m, \tau)$ that expresses the number of time steps machine $m$ needs to execute task $\tau \in \mathcal{T}_m$. By linking production time, task, and machine, it is possible to have different production times for the same task on different machines. In the skill based programming scenario, this could be caused by different implementations of the same skill on different machines. Like for example done in Brucker et al. (2006), we assume that each machine has an output buffer where the goods are stored after it finished a task.

The sequence of the tasks in an order is generally free and only restricted by the dependencies between the tasks. Thereby, the freedom of choice is constrained to implementable production sequences. Notice that only the tasks directly necessary to start some specific task need to be specified. Indirect dependencies through a chain of other tasks in between need not be given explicitly and can easily be determined, which we show in Section 4.

The goal of our scenario is to find the best schedule possible for arbitrarily arriving orders with the available machine pool. The orders compete for the machines, which can be seen as shared resources. For determining the favorable production sequences according to economic criteria, a cost function needs to be defined which has to be optimized. Depending on the production scenario, the cost function might hold material cost, storage cost, energy cost etc. and is not further specified here.

The problem at hand can be characterized as a flexible job shop problem (Kim et al., 2003), which offers three types of flexibility: Operation flexibility describes the possibility to perform the same task on different machines, sequencing flexibility means that the sequence of tasks in an order can be altered and processing flexibility allows to choose between different tasks, or sequences of tasks, which can be executed interchangeably. With the methods introduced here, only operation flexibility and sequencing flexibility can be implemented, since we formulate production plans, in which the set of tasks that have to be fulfilled is fixed and every task needs to be executed to fulfill the order. For reasons of clearer presentation we stick to this restriction, although it could be relaxed with an extension of the presented algorithms, which would enable the realization of processing flexibility.

Example

With a small running example, we will exemplify the methods introduced in this and the next sections. Our exemplary setup consists of two machines $m_1$ and $m_2$, which can perform five different tasks $\tau_1, ..., \tau_5$. The specifications of the machines are:

$$T_{m_1} = \{\tau_1, \tau_2, \tau_3\}$$
$$T_{m_2} = \{\tau_3, \tau_4, \tau_5\}$$

$$k_P(m_1, \tau_1) = k_P(m_1, \tau_2) = 1, \quad k_P(m_1, \tau_3) = 3$$
$$k_P(m_2, \tau_3) = k_P(m_2, \tau_4) = k_P(m_2, \tau_5) = 1,$$
The dependencies between the tasks are, that task \( \tau_1 \) needs to be finished before any other tasks is started and that task \( \tau_3 \) needs to be finished before Task \( \tau_5 \). The production goal is to complete three orders \( o_1, o_2 \) and \( o_3 \):

\[ T_{o_1} = \{ \tau_1, \tau_2, \tau_3 \} \quad T_{o_2} = \{ \tau_1, \tau_3, \tau_4 \} \quad T_{o_3} = \{ \tau_1, \tau_3, \tau_5 \} \]

4. FORMULATION OF THE PETRI NET MODEL

From this definition of the production problem, a Petri Net model will be generated automatically with the algorithms presented in this section. The resulting Petri Net model allows every possible execution sequence. In the Petri Net representation of the manufacturing system, the state vector \( x(k) \) represents the current status of the system at time \( k \). The controlled transitions are the manufacturing decisions to be taken and represented in the incidence matrix \( B \). Changes of the state that follow directly and without external influence are modeled as transitions in the independent part of the Petri Net and handled in the matrix \( A \).

In order to document the meaning of the places and transition, identifiers are introduced. Besides being helpful to keep the Petri Net understandable, they are used to treat elements of the Petri Net in the same way, if they have the same role in the manufacturing system. This is for example done when determining which transitions are assigned to the independent part of the Petri Net. Independent of their identifier, all places and transitions can be handled as usual. The identifiers hold information on the machine, task and order the respective place or transition belongs to. Additionally, a classifying character \( \sigma \in \Sigma = \{ S, F, P, B, I, N, C \} \) indicates the production-related meaning of the corresponding element of the Petri Net. We identified the following classifications: \( S \) relates to „start“, \( F \) to „finish“, \( P \) to „production“, \( B \) to „buffer“, \( I \) to „idle“, \( N \) to „necessary“, and \( C \) to „completed“. The algorithms presented in the following build upon this exact set of classifiers, which allow to represent quite diverse manufacturing scenarios. Nevertheless, further classifiers are possible and could enhance the representable scenarios, but would require specific consideration in the presented algorithms.

Every place corresponds to one machine \( m \in \mathcal{M} \), one task \( \tau \in \mathcal{T} \), and one order \( o \in \mathcal{O} \). Therefore, the identifier of a place is a tuple of the form \( (m, \tau, o, \sigma) \) indicating that its marking holds information on the execution of task \( \tau \) of order \( o \) on machine \( m \). Every transition connects two places and therefore its identifier needs to hold two machines \( m, m' \in \mathcal{M} \) and two task \( \tau, \tau' \in \mathcal{T} \). Since the tasks being executed belong to a specific order and since different orders must not be mixed, the identifiers of the transitions only comprise one order \( o \in \mathcal{O} \). Hence, the identifier of a transition is a tuple of the form \( (m, m', \tau, \tau', o, \sigma) \).

In the Petri Net representation, the production times are modeled as holding times of the production places, which can be transformed into a chain of production places. Each element of this chain corresponds to the time that already can be transformed into a chain of production places. Each modeled as holding times of the production places, which requires specific consideration in the presented algorithms. But would require specific consideration in the following build upon this exact algorithms presented in the following.

The orders to be produced are introduced in the second for-loop. For every order a starting place \( p_{(0,0,S)} \) is created holding one token which represents the product that’s being produced and is passed from machine to machine as the product evolves. For every task of every order, the machine on which it will be executed is not predefined and therefore all the possibilities are introduced with the production places \( p_{(\ldots, P_1)}, \ldots, p_{(\ldots, P_{f(m,o)})} \) and the buffer places \( p_{(\ldots, B)} \). For example, a token in the place \( p_{(m,\tau,o,B)} \) means that the product, which shall be produced in order \( o \) is stored in the output buffer of machine \( m \) after the task \( \tau \) was finished. The necessity places \( p_{(0, \ldots, N)} \) and the completion places \( p_{(0, \ldots, C)} \) indicate whether a task of some order still needs to be executed or that it was already completed, respectively. This is needed to keep the status of an order up-to-date and to determine which subsequent tasks are ready to be started.

In order to prevent dead transitions, which can never fire, we developed the recursive function \( f_N(\tau, \tau') \). It determines through Algorithm 2 whether a task \( \tau \) must necessarily be completed before a task \( \tau' \) can be started, and thereby resolves the indirect dependencies between the tasks. It is evaluated in Algorithm 3 right before the starting transitions and their corresponding arcs are created, and thereby prevents dead starting transitions. Note that the function \( f_N \) is not needed to get a valid Petri Net model of the factory, but only prevents dead transitions and thereby reduces the problem size, i.e., the number of transitions, without having any effect on the possible production sequences.

In Algorithm 3, first, transitions and arcs are introduced which allow to execute every task of every order in an
Algorithm 3. Create Transitions and Arcs

```
for every order \( o \in \mathcal{O} \) do
  for every task \( \tau \in \mathcal{T}_o \) do
    if task \( \tau \) does not require any prior task then
      Create a starting transition \( t_{(\ldots, P)} \) with the input arcs
      \( (P(0, \tau, o, S), t_{(0, m, 0, \tau, o, S)}) \),
      \( (P(0, 0, 0, 1), t_{(0, m, 0, \tau, o, S)}) \) and
      \( (P(0, 0, 0, S), t_{(0, m, 0, \tau, o, S)}) \) and the output arc
      \( (t_{(0, m, 0, \tau, o, P_1)}, P(\tau, o, P_1)) \) for
    end
  end
end
```

```
Algorithm 1. Create Places

Function \( f_N(\tau, \tau') : \text{bool} \)
for every Task \( \tau \) being necessary to start Task \( \tau' \) do
  if \( \tau = \tau' \) then
    return 1
  else
    return \( f_N(\tau, \tau') \)
end
return 0
```

```
for every machine \( m \in M \) do
  Create an idle place \( p_{(m, 0, 0, 1)} \) holding one token
end
for every order \( o \in \mathcal{O} \) do
  Create the starting place \( p_{(0, o, 0, S)} \) holding one token
  for every task \( \tau \in \mathcal{T}_o \) do
    Create an unmarked completion place \( p_{(0, \tau, o, C)} \)
    Create a necessity place \( p_{(0, \tau, o, N)} \) holding one token
    for every machine \( m \in M \) do
      if task \( \tau \) \( \in \mathcal{T}_m \) then
        Create \( k_p(m, \tau) \) unmarked production places
        \( p_{(m, o, P_1)} \), \ldots, \( p_{(m, o, P_k(m, \tau))} \)
        Create an unmarked buffer place \( p_{(m, \tau, o, B)} \)
      end
    end
  end
end
```

```
Example

In the running example, 53 places are created. Those are two idle places, one for every machine, three starting places, one for every order, nine necessity and completion places, one of each kind for every task of every order, twelve buffer places, one for every task of every order on every machine that is able to execute it, and 18 production places. In Algorithm 3, 39 transitions are created, 14 for the orders \( o_1 \) and \( o_2 \) and eleven for the order \( o_3 \). In order \( o_3 \) only the machine on which the task \( \tau_3 \) is executed can be chosen, as depicted in Figure 3.

5. REPRESENTATION OF THE PETRI NET MODEL
IN STATE SPACE

For the automatically generated Petri Net, the state space representation can be generated automatically as well. First, the initial state \( \varphi^0 \) is determined, which simply holds the initial markings of the places. In the second step, the matrix \( A \) is initialized as identity matrix. Since the machines move the parts to their output buffer automatically after
```
finishing transitions, the diagonal entry corresponding to the input-place $p_{(...,P_k,m(\tau,,m))}$ is removed as well, and one is introduced in the same column at the position of the output-places $p_{(...,B)}$, $p_{(...,I)}$ and $p_{(...,C)}$. By this mechanism, every token which enters $p_{(...,P_k)}$ is moved once further in every time step and tokens are added to the places $p_{(...,B)}$, $p_{(...,I)}$ and $p_{(...,C)}$ after the last production step, which is $kP(m,\tau)$ time steps later.

The starting transitions are handled in the matrix $B$, since starting a production process on a specific machine is a conscious decision. The matrix $B$ is created by adding a column for every starting transition, removing one token from the idle place $p_{(...,I)}$, the necessity place $p_{(...,N)}$, and the buffer place $p_{(...,B)}$ by introducing $-1$ in the corresponding row, and adding one token to the first production place through entering $+1$ in the row corresponding to $p_{(...,P_k)}$.

If the identifiers of the places and transitions are stacked in vectors $x^{ID}$ and $u^{ID}$ according to the generation of the state space description, it is possible to reconstruct the Petri Net from which the state space description was created and update its marking at time $k$ by evaluating $x(k)$. Thereby, also the initial description of the manufacturing system can be updated according to its evolution.

**Example**

The state space description of the overall Petri Net of the running example has 53 states, each representing the marking of a place of the Petri Net, and 21 controlled inputs, which correspond to the starting transitions. For the small excerpt which is depicted in Figure 3, the state space description is

$$x(k+1) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \end{bmatrix} x(k) + \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \end{bmatrix} u(k),$$

with the identifiers

$$x^{ID} = \begin{bmatrix} m_1,\tau_1,o_3,P_1 \\ m_1,\tau_1,o_3,B \\ m_1,\tau_3,o_3,P_1 \\ m_1,\tau_3,o_3,B \\ m_2,\tau_2,o_3,P_1 \\ m_2,\tau_2,o_3,B \\ m_2,\tau_5,o_3,P_1 \\ m_2,\tau_5,o_3,B \end{bmatrix}$$

and $u^{ID} = \begin{bmatrix} 0, m_1, \tau_1, o_3, S \\ m_1, m_1, \tau_1, o_3, S \\ m_1, m_2, \tau_3, o_3, S \\ m_2, m_2, \tau_3, o_3, S \end{bmatrix}$.

Note that equation (3) and the identifiers (4) only correspond to the excerpt of the Petri Net depicted in Figure 3, where places corresponding to the orders $o_1$ and $o_2$ as well as the idle, starting, necessity and completion places are omitted. It can be seen that the dimension of the state space is rather large even for relatively small systems.

6. DISCUSSION AND OUTLOOK

With the goal of controlling a flexible production plant on which individual orders are produced, we developed a formalism to automatically, at the push of a button, deduce a linear discrete time system from a coarse description of the production problem. Based on the discrete time system and a cost function which assigns cost to the inputs and states of the system, the production plan can be optimized. By the abstract formulation of the production orders, those already comprise the flexibility of choosing how to fulfill them. This is on the one hand the sequence in which the production steps are executed, and on the other hand the way in which they are implemented on different manufacturing units. In our framework, the first type of flexibility is explicitly considered, whereas the second type is abstracted by different execution times of the same task on different machines, which connects right to the idea of skill based programming, where the implementation of a task on a specific machine is abstracted on a similar level. In an intermediate step, a Petri Net description of the production problem is generated, from which the discrete time state space model is built. To distinguish automatic executions from conscious manufacturing decisions, we distinguish two classes of Petri Net transitions, the independent transitions and the controlled ones. Through this distinction, the
firing count vector $u$ combined with the incidence matrix $B$ only holds the active decisions and their effects on the system, whereas the autonomous evolution of the production process is captured in the matrix $A$, which we introduced in the Petri Net dynamics.

The model presented here is far from covering every aspect of production systems, which is intended to keep the complexity manageable. However, the model holds the possibility to be expanded in many ways and incorporate further aspects like for example cyclic production, shared resources, buffer sizes, setup times for the machines, or more complex dependencies between tasks by introducing extensions of the Petri Net. Since real production systems are way bigger than the example shown here, they will result in large Petri Nets and high dimensional state space descriptions. In Bachmann-Landau notation, the number of Places in the Petri Net and the dimension of the state vector $x$ is of order $O(n_2^m n_0 n_n)$, the number of transitions and the dimension of the input vector $u$ is of order $O(n_2^m n_2 n_n)$, and the number of arcs and the number of non-zero entries in $A$, $B_m$ and $B_p$ combined is of order $O(n_3^m n_2 n_n)$. Furthermore, the dimension of the state vector increases by choosing a smaller discretization step size. Thereby, a compromise between the dimension of the problem setup and the achievable precision of the resulting schedule is necessary. In order to keep the problem computationally tractable, abstraction and distribution techniques need to be developed and investigated.

In the presented setup, the decomposition of the orders into tasks and the formulation of the capabilities of the machines as skills, as well as the formulation of a meaningful cost function, is omitted and left to the operator of the manufacturing system. Those tasks are not trivial and have a great influence on the usefulness of the model and the schedules which can be obtained.

Most importantly, the presented modeling framework only yields a benefit, if its capabilities are exploited. In particular the state space description can be used to apply established methods from systems and control theory, which could open new opportunities by introducing feedback and online optimization. Since the constraint that the number of tokens in a place needs to be non-negative has to be met, Model Predictive Control is very promising, since it can explicitly take the constraints into account.

REFERENCES


