Automated Determining of Manufacturing Properties and Their Evolutionary Changes from Event Traces

Jan Ladiges¹ · Alexander Fay¹ · Winfried Lamersdorf²

Received: 20 January 2016 / Revised: 20 May 2016 / Accepted: 26 May 2016 / Published online: 10 June 2016
© Springer Science+Business Media Singapore 2016

Abstract Production plants are usually kept in operation for several decades. During this long operational phase operation requirements and other production conditions change frequently. Accordingly, the plants have to be adjusted in behavior and/or structure by adapting software and physics of the plant to avoid degeneration. Unfortunately, in industrial practice, changes, especially smaller ones, are often performed ad-hoc without appropriate adaptation of formal models or documentation. As a consequence, knowledge about the process is only implicitly available and an evaluation of performed changes is often omitted, resulting in sub-optimal production performance. Present research approaches to overcome these deficiencies usually concentrate on (a) manual modelling with manual or automatic analysis on a high level of abstraction; or (b) on automatic model generation from observations without lifting gathered knowledge to easy interpretable indicators. The approach presented in this paper combines both methods (a) and (b) by learning models from observation of input / output signals of the production plant's control system. Semantics are added by using a priori information modelling which is less tedious compared to modelling the process itself. The learned models are used to automatically detect changes by continuously comparing their behavior with real plant behavior during operation as well as to evaluate performed changes. An analysis of the models results in high-level property values such

as key performance indicators or flexibility measures of the production system.

Keywords Plant evolution · Manufacturing plants · Information modelling · Model learning · Process analysis

1 Introduction

1.1 Motivation

Due to high investment costs, production plants are usually kept in operation for many years up to several decades [55]. During this long operational phase requirements and environmental conditions change frequently [47]. To keep the plants profitable despite of high operating and maintenance costs (i.e. to avoid degeneration), the plants have to be adapted in accordance with changed production conditions and market demands [37]. In other words, production plants undergo an evolution process during their operational phase. Moreover, evolutionary changes should be seen as a part of the plant's life-cycle rather than as an exceptional activity [42,47].

The recommended way of applying changes on a technical system is to carry out a re-engineering process by (1) formalizing new or changed requirements, (2) designing a system solution (by changing the existing one), preferably in a model-based approach (3) verifying this solution against the requirements, and (4) finally implementing the desired changes on the real system [8].

However, formalizing requirements and pre-evaluating changes before implementation is often omitted in practice, instead, changes due to informal requirements are directly implemented [8,15,47]. The main reason for this course of action is high time and cost pressure [8]. Especially small unanticipated changes implemented within short periods of

✉ Jan Ladiges
ladiges@hsu-hh.de

¹ Automation Technology Institute,
Helmut-Schmidt-University Hamburg, Holstenhofweg 85,
22043 Hamburg, Germany

² Distributed Systems and Information Systems, University of
Hamburg, Vogt-Kölln-Straße 30, 22527 Hamburg, Germany

time usually lack appropriate documentation and evaluation [47]. Companies often inadequately evaluate manufacturing properties after changes have been carried out [8]. This especially holds for evaluations in terms of non-functional properties such as, for example, performance or flexibility measures.

Obviously, this practice entails risks and disadvantages. The changed behavior is not properly documented and, as a consequence, process knowledge is not explicitly available [22]. Documented plant behavior does not completely cover the real plant behavior and may even contradict the actual system implementation. To close a-posteriori the resulting *knowledge gap* between explicitly (formally) documented plant behavior and real plant behavior is usually an error prone activity demanding high effort.

Moreover, production plants are highly complex. Different disciplines are involved in the (re-)engineering process, namely mechanical engineering, electrical engineering and software engineering [46]. In accordance, evaluating changed behavior includes evaluating the complex interrelationships between those disciplines [9]. Furthermore, changes in one discipline can affect others [47]. For this reason, unintended cross-discipline property shifts may arise when changes are implemented [53]. Detecting and evaluating such unwanted side-effects can be very complicated and time consuming [47], especially under consideration of the aforementioned deficiencies of undocumented changes. As a result, non-functional properties may be unsatisfactory distinctive, and unnecessary weaknesses are even not recognized by the staff. Therefore, ensuring consistency between documentation and behavior for assuring quality belongs to the main challenges when considering evolving production plants [48].

1.2 Overview of the Approach

The considered systems are manufacturing plants and machines producing discrete goods. In contrast to process plants which deal with continuous product flows and variables, these systems are characterized by discrete variables [16]. Accordingly, the automation systems controlling the plants deal with binary input and output signals and the state of the controlled system can be modelled as a discrete event system with sufficient meaningfulness [2].

The approach proposed in this paper aims at supporting the automatic documentation of behavioral changes of these systems by exploiting event traces occurring in automation systems of manufacturing plants. To do so, traces of the binary input and output signals which can be observed between the sensors and actuators on the one hand and the controllers [(in most cases, Programmable Logic Controllers (PLCs))], henceforth called *I/O-traces*, are recorded.

Models reflecting different aspects of the plant behavior are automatically generated from these traces. Internal vari-

ables of the controllers are explicitly not considered because program execution may be influenced otherwise and acceptance of the approach is expected to be higher if no insight into the control software is required. The learned models are called *knowledge models* since they carry knowledge about the underlying automated technical process.

In order to generate specific models and to analyze those models regarding high-level properties, recorded signals are embedded in an information model which provides the semantics of the signals. This allows interpreting the I/O-traces and generating models with high abstract semantics. A meta-model including commonly needed base semantics is provided to keep the effort for creating an information model as small as possible. Behavior changes are detected automatically by online monitoring and comparing the observed I/O-traces with the model behavior. An automatic analysis of the learned models allows deriving actual values of non-functional properties as a basis for evaluating changes in the plant behavior. In summary, the approach tries to lift low-level semantics such as simple *true*- or *false* states of sensor and actuator signals to high abstract property values that can also be understood by the plant management.

The remainder of the paper is structured as follows: Sect. 2 discusses the state of the art in manufacturing system analysis and automated model generation in order to explain the trade-off between modelling effort and knowledge abstraction. Section 3 examines the systems under considerations for the presented approach. Assumptions are defined and the properties of interest are discussed to justify the decisions taken in the approach. Subsequently, a case study plant is shortly introduced which is used as a running example throughout this contribution. The approach is explained in detail in Sect. 5 and it is shown how the example properties introduced in Sect. 3 are determined in the case study. The paper concludes with a summary and some outlook on future work.

2 Manufacturing Analysis and Automated Model Generation: State of the Art

The analysis of production processes is in general based on an analysis of models which reflect the behavior of the manufacturing plants. Different models can be used, either during the design, implementation, and test phase of a production system (see Sect. 2.1), or during the operation phase of a plant, for the diagnosis of systems and their behavior [46], see Sect. 2.2.

2.1 Model-Driven Engineering and Model-Driven Testing

One approach suitable to design and pre-verify automated production systems is the concept of *model driven engineer-*

ing (MDE). MDE is widely used in software engineering, and several approaches employ MDE for the plant automation domain [50]. Usually, MDE uses both structural and behavioral models. Different modelling languages have been used for the MDE of automated production systems, e.g. general modelling techniques like the *unified modelling language* (UML) [41] or languages more tailored for systems engineering as the *systems modelling language* (SysML) [50]. Often, these approaches result in automatically generated control code [7, 41, 49].

Modelling languages as UML and SysML allow for an abstract view on the system solution during engineering and result in proper documentation artefacts. But, on the other hand, these MDE approaches have in common that an analysis of the modelled system solution is highly dependent on the model granularity. Relations to requirement fulfillment are to be modelled manually and correctness of these relationships is usually dependent on experience and modelling skills of the modeler. Apart from structural compatibility, automatic verification in the field of technical systems and especially manufacturing systems is rather rare [17]. One approach trying to automatically verify functional conformance is presented in [32]. The authors use model-checking for verifying compatibility of a system model with its corresponding requirements model. However, results are still strongly related to the model granularity. As a fact, modelling effort highly increases with rising model granularity. In addition, detailed knowledge about system parts and their behavior is needed to create an accurate model.

Other system analysis techniques are based on pure behavior modelling with *automata*, *Petri nets*, and other state or event-based modelling languages. They are suitable for requirement analysis, specification, rough and detail design, testing, simulation, as well as formal analysis and verification [44]. Petri nets, for example, are a common modelling language for system analysis [12, 35]. Different formal Petri net definitions have been developed and used for MDE such as e.g. *High-level Petri nets* [24]. It has also been shown that Petri nets are a suitable modelling language for the analysis of high abstract properties of production plants, such as their flexibilities [5, 6].

However, model based analysis and simulation only verifies models of the systems but not the systems themselves. Often assumptions about properties have to be made which may result in erroneous models which render the verification less useful [20]. Accordingly, all assumptions made during modelling have to be true to verify the real system. Furthermore, the system implementation has to be done correctly. Implementation errors, e.g. during commissioning of the plant, are not covered by the models and therefore not detected during verification [29].

Verifying conformance of implemented software with its specification is tackled by the field of *software testing*. Differ-

ent approaches from software engineering have been adapted to the automation domain [18]. In *test-first development*, test cases are defined prior or in parallel to the system design in order to test the control software after implementation [52]. Test cases usually represent requirements and, therefore, test execution correlates with requirement verification. However, defining the test cases demands expert knowledge and is, consequentially, error prone and requires high effort. The idea of *model-based testing* (MBT) tries to reduce the effort of test case generation [34]. Here, formal behavior specifications, (e.g. state machines) are used to automatically generate test cases. Other approaches generate formal specifications from test-cases (cf. [3, 43]). However, these approaches also demand manual modelling either of test cases or specification/requirement models. Similar to MDE, the quality of testing is highly dependent on the modelling skills and experience of the engineer creating models or test cases. In addition, testing is restricted to the software. The interrelated behavior of software and physics cannot be tested following this approach [29].

To summarize, all approaches discussed above have in common that manual modelling effort is needed. This demands modelling experts with detailed knowledge about the system to be implemented and its behavior. Assumptions have to be made which may be wrong due to either insufficient knowledge about the system and its interrelated behavior or faulty implementation. Additionally, as discussed in Sect. 1, formal models are often not created or adapted in practice due to high time and/or cost pressure, especially when applying ad-hoc changes.

2.2 Model-Based Diagnosis During Operation

Detecting faults in the behavior of technical systems during runtime is the topic of *fault detection and isolation* (FDI). Here supervision is used to compare signal models or process models with actual system behavior. These models may be continuous models, discrete event models, or hybrid models (cf. [19, 23, 45]). Approaches use predefined models (white-box approach), or behavior models learned from process observation (black-box approach) where no a-priori model is needed. Approaches for continuous systems use machine learning techniques to e.g. parametrize linear or non-linear differential equations, train neural networks, estimate initial states for observers, or estimate parameters for fuzzy-models depending on the type of the observed process and the amount of available structural information, cf. [1, 23]. Some approaches learn hybrid automata to capture system behavior which comprises of both discrete and continuous system dynamics, cf. [36]. For discrete event systems, most learning algorithms generate automata (cf. [38]) or Petri nets (cf. [31]) from the I/O-traces of the plant.

After learning, process behavior is compared to model behavior during runtime. Whenever a deviation is detected, it is considered as a fault. Manufacturing processes are usually highly concurrent, resulting in a huge amount of possible states. As a result, learned models tend to be highly complex, and a diagnosis on a high-level of abstraction apart from the signal level seems to be difficult. One approach trying to reduce model complexity is to learn partial models. The algorithm presented in [38], for example, automatically divides the gathered data into subsets to create partial automata of the system behavior. However, an interpretation on property level is still rarely possible since no semantics of the signals are known. Another approach is to use a-priori information. This information has to be kept to a minimum to not highly increase time and effort consumption. In [4], the events which occur in I/O-traces are manually assigned to different processes. This includes modelling in advance which events consume or release resources for these processes which later allows an interpretation of generated models and detected anomalies on a higher level of abstraction. However, specific events are needed, which may be not (externally) available in the automation system.

Our approach also uses a priori information to lift the behavior models acquired from I/O-traces to a higher level of abstraction. But only events are considered which are externally available, i.e. can be collected from the automated production system without additional effort. And, in contrast to [4], no specific events are needed. It is rather assumed that typical sensor and actuators are used and the signal semantics can be assigned from a given meta-model. Learning algorithms and analysis algorithms are defined on that meta-model. The approach will be described in Sect. 5.

3 Assumptions and Preliminary Examination of Considered Systems

The main purpose of the outlined approach is to automatically learn models and subsequently analyze these models regarding typical non-functional properties of interest. As it is not possible to find algorithms and methods for learning and analysis of all kinds of manufacturing plants, and general properties of interest also differ when comparing, for example, continuous plants and discrete parts manufacturing, we will state some assumptions and restrictions for the considered systems (Sect. 3.1). However, the restrictions cover most discrete goods productions and, therefore, the range of considered systems is still enormously wide. Subsequently, we discuss the high level properties of interest for this approach suitable to evaluate performance of the considered systems (Sect. 3.2).

3.1 Assumptions and Restrictions

One way to classify manufacturing processes is to divide them into discrete parts processes and continuous processes (cf. [11]). The systems considered here are discrete parts processes with discrete dynamics, fully controlled by programmable logic controllers (PLCs). In other words, the subject of the manufacturing systems considered here are discrete goods (workpieces) which are automatically transported by a material flow system and processed by a set of machines. The considered systems may produce various products. The different workpiece types can be distinguished by measurements, e.g. by measuring color, shape, or material.

In addition, the system behavior can be modelled in terms of a *discrete event system* (DES). This holds for most discrete parts manufacturing plants [16] such as *flow shop systems*, *automatic transfer line systems*, *job shop systems*, *flexible manufacturing systems*, and *assembly systems* [35]. Each discrete state can be represented as a set of binary sensor and actuator signal values. The sensor signals are input signals of one or more PLCs which cyclically and deterministically process the sensor events and calculate binary actuator signals accordingly. Typical sensor signals are workpiece detection signals (e.g. stemming from lightbarriers) and machine movement signals (e.g. from position indicator switches). Typical actuator signals are e.g. motor on/off (e.g. to control a conveyor belt or a machine tool).

Since the approach is based on model learning, it is assumed that the externally visible behavior (characterized by the I/O-traces, i.e. the chain of events) has been observed completely and fault-free during an observation phase. Further, it is assumed that the behavior has been observed by a subscription mechanism, i.e. the observed behavior is provided in a trace of events, and each event describes the change of an I/O-signal and has a timestamp¹ assigned.

3.2 Properties of Interest

The properties to be automatically determined should be interpretable by the plant staff, especially the plant management and allow an evaluation of the production. Therefore, the properties of interest for this approach are non-functional properties generally defined for production plants. Examples for such properties are different KPIs like the throughput rate, the allocation ratio, or the utilization efficiency as defined in ISO 22400-2 [13]. Other properties like different measures for flexibilities as given e.g. in [40] are also considered. Obviously, only those properties can be determined which are operationalized in such a way that they are reflected in the

¹ Such data can easily be obtained e.g. by using the commonly used OPC technology (cf. [30]).

I/O-traces [26]. In the following, we will discuss exemplarily two properties of sufficient complexity.

3.2.1 Throughput Rate

The throughput rate is, according to ISO 22400 [13], an important index for the production since it directly describes the outcome of a manufacturing process by indicating the produced quantity per unit time. The throughput rate is calculated as the ratio of *produced quantity* and the *actual order execution time*. The order execution time is defined as the time difference between start time and end time of a production order. Since orders are usually processed in manufacturing execution systems and, therefore, cannot be extracted from I/O-traces, we will simplify the actual order execution time as the time difference between the first workpiece entering the plant and the last one leaving it during the observation. Since production of different product types may take different durations, this information is only valuable in combination with the product mix that has been produced. Additional important information for evaluation of the throughput rate is the mean production time needed for each product type. Accordingly, the following characteristics are needed to calculate and evaluate the throughput rate:

- Time needed for production of all products in a certain observation phase
- Number of produced goods (per product type)
- Mean time needed for production of products (per product type)

3.2.2 Process Flexibility

A more complex property is the process flexibility which is defined as related to “the set of part types that the system can produce without major setups” [40]. Proposed measurements for process flexibility concentrate on the number of parts the system can (or cannot) produce (respectively the range of treatable shapes, sizes, etc.) or the effort to change between different parts. The given definition correlates with the definition of *product mix flexibility response* by Wahab which is given as “the ability to change between current products” and specifically the ease to do it [51]. Wahab also gives a metric for measuring the product mix flexibility response of a system:

$$SPMFR = \frac{1}{m} \sum_{j=1}^m \sum_{k=1}^n \sum_{l=k}^n p_{kj} a_{kl} \quad (1)$$

Here m denotes the number of machines in the plant, n the number of distinct product types, and p_{kj} is the probability

that product k will be processed at machine j . Wahab also gives a metric for p_{kj} depending on the number of operations the machine can perform and the efficiency of the operation. However, since our approach is based on observations, we can also define p_{kj} as the ratio of number of produced products of type k processed at machine j to the overall number of produced products of type k .

The parameter a_{kl} is defined as

$$a_{kl} = \frac{|T_k \cap T_l|}{|T_k \cup T_l|} \quad (2)$$

with T_k and T_l as the sets of tools needed for the production of product k and product l , respectively. Which tools are used is probably not reflected in I/O-traces. Therefore, tools are replaced by operations.

In summary, the following basic properties are to be determined for this definition of process flexibility:

- Number of processing machines in the plant
- Assignment of product types to processing machines
- Set of operations needed to produce each product type

4 Exemplary Case Study

For a better understanding of the approach, we shortly introduce here a case study which has been carried out to evaluate the approach on a laboratory plant at the Automation Technology Institute of the Helmut-Schmidt-University in Hamburg (Fig. 1, upper part). Results of the case study are provided in Sect. 5 to exemplify the approach.

The plant is divided into five subsystems, each controlled by its own PLC, with altogether 150 binary sensor and actuator signals. It consists of 12 conveyor belts, 8 turntables, and 3 different machine tools. The plant is able to process different workpiece types, identified by reed contacts at the plant entrance. In the case study, the plant has been run to deal with six workpieces of two different types, Type 1 and Type 2. The three workpieces of Type 1 have been sorted out after entering the plant by means of the first turntable. The three workpieces of Type 2 have been processed by the machine tools. One single production cycle is shown in the lower part of Fig. 1 by an abstract state machine. Since the workpieces do not pass any position in the plant more than once during production, there are no equivalent states or transitions in the automaton. Accordingly, without formal proof, the automaton is a minimal realization of one single production cycle. Note that different workpieces can be processed at the same time and, accordingly, different production cycles can run in parallel. However, each subsystem (SS1–SS5) can carry only one workpiece per time.

Fig. 1 Case study laboratory plant. *Up* Plant top view—conveyor belts, turntables, machines. *Down* State machine of one production cycle

All events generated by the signals during operation have been recorded by using an OPC data logger connected to the plant I/Os via Ethernet communication. All algorithms have been implemented in *Matlab* and can be provided on demand.

5 Approach for Automated Change Evaluation from I/O-Event Traces

In this section, we will present our approach in detail. The generic concept framework is discussed in the following

paragraphs. Subsequently, the methods required to implement the concept are presented.

The main focus of the approach is on lifting raw data (the I/O events) to knowledge in terms of abstract property values (see Fig. 2). Manual effort has to be kept to a minimum in order to meet the practical boundary conditions discussed in Sect. 1. However, as known from knowledge management, data can just be raised to information by adding a meaning to the data, respectively by adding semantics (cf. [33]). Therefore, the approach follows the idea of injecting semantics by information modelling. To keep the manual effort low, the required semantics are restricted to the plant topology and

Fig. 2 Generic concept to determine property values

the types of recorded signals. In addition, a meta-model for the signal types is given (see Sect. 5.1).

Based on these semantics, the data is automatically separated to reduce model complexity and to be able to depict specific aspects of the process. Subsequently, the pre-processed data is used to automatically learn knowledge models (Sect. 5.2). Many properties of interest can be derived from the material flow within the manufacturing plant. Therefore, one type of knowledge models depicts the material flow including the number of products treated during production as well as their routing and all timing information. In addition, for each technical resource in the plant, one model is learned reflecting the resource's behavior. The whole set of learned models is used to detect changes during runtime. To do so, anomaly detection methods are used (see Sect. 5.3). Finally, the models are automatically analyzed to calculate property values (Sect. 5.4). Here, the semantics given in the information model are exploited in order to correctly interpret the models regarding the properties to be analyzed.

5.1 Information Modelling

The purpose of the information model is to provide semantics needed for separating the data traces, learning models reflecting specific aspects (e.g. the material flow), and ana-

Fig. 3 Meta model of the information model (left) and instance example of the case study (right)

lyzing the models. The structure of the model expresses the plant topology. On the lowest level of this hierarchical model, all I/O signals are assigned to the technical resources, e.g. a conveyor, a turntable, or a machine tool. Further, each signal is provided with a *SignalType* including the meaning of the signal. The meta-model of the information model is given in Fig. 3 on the left side. An example of such an information model is given on the right side of Fig. 3, showing partially the information model of the case study plant. Here, three hierarchical levels are given in the topology model. On the top level the plant is modeled as a whole. The second level consists of the subsystems and the lowest level consists of the single technical resources. The example shows machine tool *M3* in subsystem *SS5* and the corresponding signal *B5_A09*. This actuator signal is for switching on and off the tool and is therefore of type *A.WPModify*.

Signal types are predefined to reduce effort in creating the information model and to define algorithms using the provided information. The most important signal types, which are used by the general algorithms and rules, are explained in Table 1. Further signal types could be added for specific analysis or to define further algorithms for learning models reflecting specific aspects not considered so far. Note that the information model has only to be adapted during plant evolution when new signals of relevant types are added, signals are removed from the plant, or if their semantics are changed.

5.2 Learning Knowledge Models

The current state of the system is given by the I/O-vector of the system which contains the state of each binary sensor and actuator signal. Each state which can be represented by the I/O vector is called a *symbol* and the set of all possible symbols (i.e. all states that have been observed) is called

Table 1 General set of important signal types

Signal type	Explanation	Example
<i>A.WPHold</i>	Actuator to hold or grasp a workpiece	Pneumatic grapper on/off
<i>A.WPModify</i>	Actuator to modify a workpiece	Drilling machine on/off
<i>A.Toolchange</i>	Actuator to change a tool in a machine	Rotate machine tool magazine
<i>A.Move</i>	Actuator to move a resource	Motor on/off to rotate a turntable
<i>A.State</i>	Actuator to change the state of a resource	Motor to move a machine
<i>S.State</i>	Sensor to describe the state of a resource	Respond of a motor on/off
<i>S.Position</i>	Sensor to describe a machine's position	Limit switch
<i>S.WPIdentify</i>	Sensor to identify workpiece types	Inductive sensor to distinguish metallic/non-metallic workpieces
<i>S.WPDetect</i>	Sensor to detect presence of a workpiece	Lightbarrier
<i>S.WPHold</i>	Sensor to indicate a resource holding a workpiece	Sensor response of a grapper holding a workpiece
<i>Auxiliary</i>	Signals to be not further considered	Actuator switching signal lamp on/off

the *alphabet* of the system. Furthermore, adding the timing information for each event results in a *timed alphabet* containing pairs of symbols and corresponding timing attributes (see [39] for further details).

Based on the recorded event traces and the semantics given in the information model, knowledge models are learned which represent the same timed alphabet as the observed system. The advantage of using added signal semantics is the possibility to divide the I/O vector of the plant into sub-vectors that only contain specific signal semantics and reflect accordingly specific parts and specific aspects of the plant. Exploiting the signal semantics can thus lift the semantic meaning of a learned model to a higher level of abstraction and the models can be analyzed on a higher semantic level.

Since Petri nets are widely used for different tasks during the life cycle of manufacturing plants and different learning algorithms already exist which create Petri nets from event traces (see Sect. 2), Petri nets are chosen as suitable modelling language for the approach. The main advantage to other description languages is the capability of efficiently depicting concurrency (cf. [28]) and, accordingly, material flow in production systems (cf. [12, 25]). Two types of models are learned. One type represents the full behavior of single technical resources. In other words, they contain and combine all I/O events of signals belonging to the corresponding technical resource. They describe the state behavior of machines (or other technical resources) and are hence called *machine state Petri nets* (MSPN). Most of the properties of interest are related to the material flow in the manufacturing system. Therefore, *material flow Petri nets* (MFPN) are also learned to capture the routing of workpieces and the corresponding time behavior. MSPN could also be modelled in another modelling language for discrete event systems. However, to keep modelling compatible, Petri nets have been chosen as base modelling language for MSPN as well as MFPN.

For each model to be learned just a subset of the recorded events is required. To automatically select the right signals, some rules are defined on the signal semantics as follows:

- (1) Exclude all signals of type *Auxiliary*
- (2) Divide the event trace according to the signal's affiliation to the topology in the information model
- (3) For MFPN learning, only consider events from signals with type *S.WPDetect* and *S.WPIdentify*

Further rules can easily be defined on the information model to extract other information. In [27], for example, it is shown how further material flow relevant information can be created by rule based combination of events from signals of type *S.WPHold* and *S.Position*.

In the following subsections, the model definitions for MSPN and MFPN are given, and the fundamental ideas of the learning algorithms are discussed. More information on these algorithms can be found in [25, 28]. How knowledge models can be represented in a flexible and with the plant co-evolving software environment has been shown in [21].

5.2.1 Machine State Petri Nets

A MSPN is a Petri net P, T, F where P is the set of places, T is the set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is the set of directed arcs between places and transitions. Each transition $t \in T$ is annotated with a set of events $\tilde{e}_j$ which is a subset of the recorded event trace E_i . Here, the index i refers to the i^{th} resource in the information model and E_i , accordingly, only contains events related to that technical resource, which can be distinguished as a result of the preprocessing steps explained above. Further, to cover a wide set of characteristics that can be extracted from the event traces each transition is annotated with a 5-tuple $d_{min}, d_{max}, \mu, \sigma, n$. The elements are:

Fig. 4 Schema of a conveyor

d_{min} : Minimum activation duration of t
 d_{max} : Maximum activation duration of t
 μ : Mean value of activation duration of t
 σ : Standard deviation of activation duration of t
 n : Number of firings of t during observation

The MSPN learning algorithm is based on the causality specification defined by Lefebvre and Leclercq [31]. Moreover, to meet the specific needs of our approach, some modifications and extensions have been made.

In a first step E_i is further preprocessed. To do so, a timing threshold $T_{thresh} = u \cdot T_{sample}$ as a multiple of the sampling time during event recording is defined. The sampling time is usually restricted by the data recorder and the communication protocol used to record the event trace. All events in E_i occurring within a time interval smaller or equal to this threshold are summarized to a combined event. This avoids an increase of model complexity due to unprecise and scattering timestamps. However, it has to be taken into account that dynamics with timing equal or less T_{thresh} are not correctly captured by the resulting model.

Subsequently, in step 2, a Petri net is generated from the event trace, following the causality specification by Lefebvre and Leclercq in [31]. The output is an incidence matrix describing a state graph Petri net.

To avoid behavior of the MSPN which contradicts the fact that a signal can just toggle between its two states (i.e. the high state and low state of a signal), further places are added to the MSPN in step 3. For each signal contained in the MSPN, two places representing its high state and low state are added. Arcs are created such that for every low event of a signal, a mark is transported from its high-state place to its low-state place and vice versa. As a result, the permissivity of the model is reduced by exploiting that binary signals can just toggle between their two states.

In a last step (4), the 5-tuples to be annotated on the transitions are calculated from the event trace.

As an example from the case study, Fig. 4 shows the schema of Conveyor 3.3, a conveyor in front of a machine tool. If the conveyor is requested, it starts transporting by activating actuator $B3_A12$ until a workpiece is recognized by sensor $B3_S09$. The conveyor stops, the workpiece is processed, and once it is finished, the conveyor starts again, and the workpiece is transported to the end of the conveyor. Table 2 shows a part of the recorded event trace for the conveyor.

Table 2 Part of the recorded eventtrace

Timestamp in ms	'Signal'	Value
388400	'B3_A12'	1
389976	'B3_S09'	1
389992	'B3_A12'	0
398088	'B3_A12'	1
398244	'B3_S09'	0
399430	'B3_A12'	0

After the outlined learning algorithm with $T_{thresh} = 60ms$ has been executed, the model as shown in Fig. 5 is generated. The left part of Fig. 5 shows the net generated after performing steps 1 and 2. Due to the combination of events which occur within T_{thresh} , the rising edge event of $B3_S09$ and the falling edge event of $B3_A12$ are combined to one event (compare line 2 and 3 in Table 2 as well as T2 in Fig. 5). The upper left part shows the places generated by step 3 and how they are connected to the transitions. Since these places have a specific meaning (step 3 above) the signal-state they describe is annotated. Note that the transitions are the same as in the left part of Fig. 5. They are just depicted anew for a better comprehension. The table in the lower right part of Fig. 5 shows the annotations of the transitions.

A MSPN can be interpreted as a stochastic timed automaton. Since the generated Petri net without the part imposing signal toggling is a state graph Petri net (i.e. there is only one token in the net and each transition has exactly one incoming and one outgoing arc) it can also directly be transformed into an automaton, cf. [12]. The part imposing signal toggling is just prohibiting some possible state transitions and enhances the state space by adding the signal states. Probabilities are given by the number of firings (n). If there is any choice in the net, the probabilities can be calculated by the relation of numbers of firings. As a consequence, all analysis algorithms for stochastic timed automata (see e.g. [54]) can also be used for MSPNs.

5.2.2 Material Flow Petri Nets

As mentioned before, many properties of interest depend on the material flow in manufacturing systems. Therefore, Petri nets reflecting the material flow shall be learned from the event trace. The resulting nets are called *material flow Petri nets* (MFPN) and depict possible routes in the system, the transportation time between any two points (sensors) in the system as well as which workpiece types take which routes. To be able to reflect all these information, MFPNs are defined as follows (for more detailed information, we refer to [25]).

A MFPN is a Petri net P, T, F where transitions are annotated with signal names (of type WP_{Detect} , see Sect. 5.1).

Fig. 5 Resulting MSPN of the conveyor

Marks in a MFPN represent workpieces. Transitions represent positions in front of a sensor and places represent positions between sensors. To analyze the transport times and cover a wide set of characteristics extractable from the event trace, each transition is annotated with one or more 6-tuples d_{min} , d_{max} , μ , σ , n , $ident$. The elements are

- d_{min} : Minimum transport duration
- d_{max} : Maximum transport duration
- μ : Mean value of transport duration
- σ : Standard deviation of transport duration
- n : Number of firings during observation

$ident$: Vector of signals identifying types of workpieces (signal type *WPIIdentify*, see Sect. 5.1)

For each different workpiece type, described by the vector $ident$,² one 6-tuple is annotated in order to be able to express the material flow for each workpiece type separately. Transportation duration means the duration a workpiece needs to pass the sensor providing the annotated signal, i.e. the time the signal *WPDetect* is “high” due to one passing workpiece. Transitions have exactly one incoming arc and one outgoing arc (i.e. MFPNs are state graph Petri nets). To be able to unambiguously depict the material flow, places can have several incoming arcs or several outgoing arcs, but not both

(see [25] for a discussion of this detail). Places and/or arcs are also annotated with the 6-tuple given above, but according to the transport duration between sensors. Whether a place or an arc is annotated depends on the number of incoming and outgoing arcs of a place: if a place has several incoming arcs, each incoming arc is annotated; if it has several outgoing arcs, the outgoing arcs are annotated; otherwise, the place itself is annotated.

Before generating the MFPN, the event trace is pre-processed as depicted in Fig. 6. First, the trace is separated according to the events’ affiliation to the equipment on the lowest level of the information model, and all non-relevant events (i.e. events of signals which are not of type *WPDetect* or *WPIIdentify*) are filtered out. Subsequently, the events are assigned to workpiece instances. That is, it has to be identified which events are triggered by the same workpieces and which are triggered by different ones. This is done by analyzing the time stability between events. If same events are triggered several times in a stable time difference, they are assigned to same workpiece instances. After IDs have been assigned for the equipment, these IDs have to be merged to “global” plant-wide IDs. This is done similarly to assigning local IDs. For each event of a technical resource, the chronologically last events of each ID are taken and first events of other resources are searched, with a stable time difference.

To show an example, the subsystems *SSI* and *SS2* of the case study are considered, as depicted in Fig. 7. Workpieces entering the plant are firstly detected by sensor *B1_S02*

² $ident$ may be a barcode or RFID number or a set of signals which identify workpieces e.g. by material, such as inductive sensors do.

Fig. 6 Process of assigning workpiece instances to events for generating MFPNs

Fig. 7 Schematic of SS1 and SS2 of the example case study

and transported forward. Reed-contact sensors $B1_S04$ and $B1_S05$ are used to identify workpiece types. Workpieces of Type 1 trigger none of the sensors, and workpieces of Type 2 trigger both. On the first turntable, the workpieces are sorted: Workpieces of Type 1 are transported upwards (triggering sensors $B2_S07$, $B2_S08$, and $B2_S03$) and are sorted out at the upper turntable. Workpieces of Type 2 are transported forward to SS3 via $B2_S09$ and $B2_S10$.

Subsequently to assigning workpiece instances to events, a place-transition chain is generated for each ID. In a next step, the chains are combined in such a way that the number of places and transitions is minimized and the net still fulfills the definition given above. As a last step, time differences are calculated and annotated, as well as the corresponding identification signals. For more details, see [25].

Executing the outlined learning algorithm delivers a MFPN as partially shown in Fig. 8. Note that only some annotations (shown in braces) are exemplarily depicted and all timings (i.e. d_{min} , d_{max} , μ and σ) are given in milliseconds. As mentioned in Sect. 4, the event trace for the case study stems from a test run in which 6 workpieces have been processed by the plant. Three of them did not trigger any identification sensor and the other three triggered $B1_S04$ as well as $B1_S05$. In the MFPN this can be seen by the different *ident* vectors $[0\ 0]$ and $[1\ 1]$. Transportation durations

(in terms of mean and standard deviation, min, or max) can be calculated from that graph. Also the distinct routing of the two workpiece types can clearly be seen at the outgoing arcs of $P6$. Moreover, it can be seen that workpieces of Type 1 (detected as $[0\ 0]$) rest longer in front of sensor $B2_S06$ than those of Type 2 (on average, 1560 instead of 260 ms). The reason is that the turntable has to turn by 90° before workpieces of Type 1 can be transported further to $B2_S07$.

5.3 Detecting Changes

The learned MSPN and MFPN models are used for anomaly detection. This is mainly done to automatically capture changes in the plant behavior. To do so, an anomaly detection engine observes the I/O-events of the system during runtime and compares the plant behavior with the behavior of the learned models. If an event has been observed that contradicts the timed alphabet of the model (i.e. it contradicts the timed alphabet of the system behavior observed to generate the models), an anomaly has been detected cf. [39]. The following anomalies can be detected with MSPNs:

- An event occurs and there is no activated transition in the models with this event annotated

Fig. 8 Partial MFPN of example case study

- An event occurs and there is an activated transition with this event annotated but the activation time is less than its annotated d_{min}
- The activation time of a transition is longer than its annotated d_{max}
- In case of several post-transitions of one place: The ratio of firings (determined by the annotated number of firings during observation, n) changes dramatically (i.e. is 10 times higher or lower as given by the annotations)

In addition, the following anomalies can be detected in the material flow with a MFPN:

- A rising edge of a signal occurs but there is no activated transition with that signal annotated
- An event occurs and there is an activated transition with this event annotated, but the activation time is less than the annotated d_{min} of its preplace
- A transition is longer activated than the annotated d_{max} of its preplace
- A falling edge occurs, but the time difference between the corresponding rising edge and the falling edge is less than d_{min} of an activated transition with this signal annotated

- A transition is activated and the rising edge of its annotated signal has been observed but more than its annotated d_{max} ago
- A workpiece has already been passing all identification sensors, but passes a transition, place, or arc without its identification code annotated

Obviously, not all changes in the plant behavior originate from intended changes. Therefore, the detected changes should be presented to an operator who is able to decide if the change is due to an intended plant modification or due to erroneous behavior. In case of intended (or acceptable) behavior changes, the models can be automatically adapted or learned anew and analyzed in order to show the consequences of the changes in terms of high-level process properties.

5.4 Analyzing the Models

The learned MSPN and MFPN models can be analyzed in two ways. Firstly, they can be validated formally by checking typical properties of discrete event models. MSPNs and MFPNs have, due to the learning algorithms used to create them, specific structures. Therefore, general statements can be made on those properties which are discussed in the following subsection. In addition, the aimed extraction of process properties can be done which is exemplarily shown subsequently.

5.4.1 Formal Analysis

Discrete event systems in general and Petri net models particularly can be analyzed regarding some common properties. To do so, standard algorithms for Petri nets and other discrete event models can be used as e.g. given in [10, 12, 54]. In case of MSPNs and MFPNs, their specific structures given by the learning algorithms allow general statements about these properties. These are discussed in the following.

MSPN As stated in Sect. 5.2.1 MSPN models are initially generated as state graph Petri nets by the learning algorithm, i.e. each transition has exactly one incoming arc and one outgoing arc. The part of the MSPN generated to impose signal toggling always adds one marked preplace and one unmarked postplace for each annotated event. As a result, every transition has always exactly as many incoming arcs as outgoing arcs. Consequently, there will be no token splitting or merging and no token generation or removal. In other words, the unity vector is always a valid place invariant. Therefore, it can generally be stated that the number of states of a MSPN is finite, i.e. their reachability graph is finite and the model is bounded and stable.

As a second important fact, it is assumed, that the observed system shows repetitive and cyclic behavior (see Sect. 3.1) which is given for real manufacturing systems and machines. Thus, the learned MSPN will also show cyclic behavior without any deadlocks. If startup behavior has been observed during learning which is not part of the cyclic behavior, the states of the startup behavior will not be reachable once the model passed those states. However all places and transitions which are part of the cyclic behavior are live and only those are of interest when analyzing the models regarding process properties.

MFPN MFPN are not cyclic. They rather describe different paths of workpieces through the system from a stage entry, (modelled as transition reflecting the first sensed position of a workpiece) to one or more end positions (modelled as places after the last sensed position, see also the example shown in Fig. 8). Every time a new workpiece enters the plant, a new token is generated. Accordingly, the net is not bounded since it can generate an infinite number of tokens. Since it also has no initial marking, resp. the initial marking is a zero vector, a MFPN cannot be reversible. On the other hand all transitions and places are always alive and there are no deadlocks possible because there is no restriction on the capacity of places.

5.4.2 Process Property Extraction

The learned knowledge models are analyzed in order to determine process and plant properties as an evaluation basis for performed changes. MSPN and MFPN allow determining a variation of properties. To do so the structure of the knowledge models as well as their behavior (i.e. the annotations) is analyzed. Also the semantics given in the information model are used to interpret the models regarding the desired properties. To show how this is implemented, the analysis of the properties *throughput rate* and *process flexibility* introduced in Sect. 3.2 is explained in the following. In addition, the property values are calculated for the case study.

Throughput Rate The throughput rate as described in Sect. 3.2.1 can be calculated by analyzing an MFPN and the filtered event trace used for generating the MFPN (see Fig. 6). To determine the overall production duration, the time difference between the first event and the last event in the trace is calculated. The filtered trace is used because it only contains events of signals which detect workpieces. Accordingly, the calculated time is the time difference between the first workpiece being detected and the last workpiece being released from the plant. For the case study, a production time of 542.6 s has been calculated.

To calculate the number of produced goods per product type, at first all places without outgoing arcs are taken and

the number of firings of their pretransitions are summed up for each product type. This procedure could calculate successfully that there have been three Type 1 workpieces ([0 0]) and 3 Type 2 workpieces ([1 1]) being produced. Out of these numbers the throughput rate could be calculated:

$$TR = \frac{6WP}{542.6s} = 39.8 \frac{WP}{h} \quad (3)$$

By summing up all mean timings of each workpiece type, the mean production duration for one product of each type can be calculated. If workpieces of a type can take several routes through the system, i.e. there are different graph branches with that workpiece type annotated, the summands have to be weighted with the number of firings n annotated and divided by the overall sum of annotated n . For the case study, the algorithm calculated a mean production duration of 65.5 s for Type 1 workpieces and 126.2 s for Type 2 workpieces.

Process Flexibility Recalling Sect. 3.2.2, the number of processing machines in the plant, their assignment to product types, and the set of operations needed to produce each product type are needed to calculate a property value for the process flexibility.

The number of processing machines can be determined by determining the number of learned MSPNs which contain at least one event of signals with type *A.WPModify* (see Sect. 5.2). In the case study these are 3 MSPNs, respectively processing machines.

To assign product types to machines the event traces are checked for invariants between true states of the signals of type *A.WPModify* and true states of sensed positions. Since a workpiece has to be detected before being processed, for each *A.WPModify* signal there is a sensed position that is true whenever the *A.WPModify*-signal is true. This is also shown on Fig. 9 which shows the MFPN of the case study example from SS1 to SS3. Here, only the annotations n and *ident* are shown for simplicity since the timing information is not necessary for calculating the process flexibility. The high state of the Signal *B3_S09* correlates with the high state of one signal with type *A.WPModify* which is contained in the MSPN of machine M1. Accordingly, it can be deduced that modification of a workpiece by the machine represented by the respective MSPN is done whenever the workpiece is at position *B3_S09*. The MFPN shows, that all three workpieces of Type 2 (*ident* = [1 1]) pass this position and that all other workpieces did not pass any position which correlates with an *A.WPModify* signal. Furthermore, it could be identified that Type 2 workpieces pass always exactly the same three processing machines.

The set of operations is estimated by the set of processing machines a workpiece passes in the MFPN. Accordingly, we

Fig. 9 Partial MFPN to calculate process flexibility

can describe the abstract set of operations as $[O_1 O_2 O_3]$ each performed by one of the machines.

From the analysis results we can determine all variables needed to calculate formula (2) and finally formula (1) for the product mix flexibility response (see Sect. 3.2.2) as follows:

Set of machines (number of MSPNs containing a signal with type A.WPModify):

$$m = 3$$

Number of product types (number of different ident-vectors in the MFPN):

$$n = 2$$

Set of operations per product type (number of processing machines passed by each product type in the MFPN):

$$T_1 = \phi$$

$$T_2 = [O_1 O_2 O_3]$$

Probability of product type k being processed on machine j (p_{kj}):

$$p_{11} = p_{12} = p_{13} = 0$$

$$p_{21} = p_{22} = p_{23} = 1$$

Inserting these values into formulas (1) and (2) results in a value of 0 for the product mix flexibility response telling that there are no operations which can be used for production of several product types.

Beside those properties discussed above, further properties can be determined by analyzing MFPNs and MSPNs. Examples are the number of routes a workpiece can take through the system by determining all possible graph branches with a given *ident* annotated, the time a value adding activity is performed by summing up all timings a signal of type A.WPModify was true, or the number of routing connections between machines.

6 Summary and Outlook

Keeping documentation up-to-date is a challenging task when considering production plants which are evolving over time. Without up-to-date formal models it is difficult, resource intensive, and error prone to analyze production processes, especially in terms of non-functional high-level properties. Approaches either consider analyzing manually created models on a high level of semantic abstraction or generating models from observations which are less interpretable. This paper presented an approach trying to glue both worlds. To do so, an information modelling approach is used to add semantics to I/O signals of PLCs controlling the plants. This requires more manual effort than learning without a priori information, as e.g. done in fault detection approaches. However, the manual effort is moderate in our approach, and the gained value is significant. We have shown how observed event traces of manufacturing systems with two state discrete input and output signals combined with an information model can be used to learn models reflecting specific aspects of the system. Particularly, we have shown how different models are learned for a laboratory case study plant. Learned models reflect the behavior of single production equipment (*machine state Petri nets*) as well as the material flow within the system

(*material flow Petri net*). Based on these models, changes in the behavior of the system can be detected. In combination with the information model, it is possible to determine actual values of high-level properties such as KPIs or flexibility measures as shown in this paper.

Future work will be on reducing the manual effort needed for the approach and further generalizing the proposed algorithms and methods. Engineering tools and documents may be a suitable source for data needed to reduce the manual effort for creating the information model. To capture other classes of production plants, such as hybrid plants or continuous process plants, further model types are needed and learning algorithms and analysis algorithms have to be developed or adapted from other approaches. Suitable learning algorithms could be those presented in [36] to generate timed hybrid automata for plants with discrete and continuous dynamics and one of the many already existing machine learning algorithms for continuous dynamics as e.g. presented in [1]. Another way could be the extraction of specific events to be generated by the continuous signal. However, it has to be examined which semantics should be given to signals with continuous dynamics such that the resulting models can be analyzed regarding high-level properties of interest. Therefore, it has to be determined which general properties are of interest for continuous and hybrid production systems. Further generalization of the approach includes considering uncertain system behavior, i.e. behavior that is not directly observable. One approach could be to generate fuzzy automata (cf. [14]). To do so, suitable learning algorithms have to be found and fuzzy analysis of the resulting models should be examined as well as anomaly detection for those models.

In addition, more complex case studies will be carried out in order to evaluate and improve current algorithms and methods. For an efficient evolution support, a fast adaptation of the knowledge models should be possible, instead of learning them again from scratch. Therefore, methods have to be found which allow an incremental adaptation of MSPNs and MFPNs.

References

1. Aldrich, C., Auret, L.: *Unsupervised process monitoring and fault diagnosis with machine learning methods*. Springer, London (2013)
2. Allen, L.V.: *Verification and Anomaly Detection for Event-Based Control of Manufacturing*. Ph.D. Thesis. University of Michigan, USA (2010)
3. Ackermann, C., Cleaveland, R., Huang, S., Ray, A., Shelton, C., Latronico, E.: Automatic requirement extraction from test cases. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Rosu, G., Sokolsky, O., Tillmann, N. (eds.) *Runtime Verification*. Springer, Berlin (2010)
4. Allen, L.V., Tilbury, D.M.: Anomaly detection using model generation for event-based systems without a preexisting formal model. *IEEE Trans. Syst. Man Cybern. A Syst. Hum.* **42**(3), 654–668 (2012). doi:[10.1109/tsmca.2011.2170418](https://doi.org/10.1109/tsmca.2011.2170418)
5. Barad, M., Sipper, D.: Flexibility in manufacturing systems: definitions and Petri net modelling. *Int. J. Prod. Res.* **26**(2), 237–248 (1988). doi:[10.1080/00207548808947856](https://doi.org/10.1080/00207548808947856)
6. Barad, M., Sipper, D.: Flexibility and types of changes in FMSs: a timed petri-nets assessment of machine flexibility. *Int. J. Adv. Manuf. Technol.* **5**(4), 292–306 (1990). doi:[10.1007/BF02601538](https://doi.org/10.1007/BF02601538)
7. Bassi, L., Secchi, C., Bonfe, M., Fantuzzi, C.: A SysML-based methodology for manufacturing machinery modeling and design. *IEEE/ASME Trans. Mech.* **16**(6), 1049–1062 (2011)
8. Bellgran, M., Säfsen, K.: *Production Development: Design and Operation of Production Systems*. Springer, London (2010)
9. Braun, S., Bartelt, C., Obermeier, M., Rausch, A., Vogel-Heuser, B.: Requirements on evolution management of product lines in automation engineering. In: *Vienna International Conference on Mathematical Modelling* (2012)
10. Cassandras, C.G., Lafortune, S.: *Introduction to Discrete Event Systems*, 2nd edn. Springer, New York (2008)
11. Chryssolouris, G.: *Manufacturing Systems: Theory and Practice*. Springer, New York (2006)
12. DiCesare, F., Harhalakis, G., Proth, J.M., Silva, M., Vernadat, F.B.: *Practice of Petri Nets in Manufacturing*. Chapman and Hall, London (1993)
13. DIS/ISO22400–2: *Automation Systems and Integration—Key Performance Indicators for Manufacturing Operations Management—Part 2: Definitions and Descriptions* (22400-2) (2012)
14. Doostfateme, M., Kremer, S.C.: New directions in fuzzy automata. *Int. J. Approx. Reason.* **38**(2005), 175–214 (2005)
15. Frey, G., Litz, L.: Formal methods in PLC programming. In: *IEEE International Conference on Systems, Man, and Cybernetics* (2000)
16. Groover, M.P.: *Automation, Production Systems, and Computer-Integrated Manufacturing*, 3rd edn. Prentice Hall Press, Upper Saddle River (2007)
17. Hackenberg, G., Campetelli, A., Legat, C., Mund, J., Teufi, S., Vogel-Heuser, B.: Formal technical process specification and verification for automated production systems. In: Amyot, D., Fonseca i Casas, Pau, Mussbacher, G. (eds.) *System Analysis and Modeling: Models and Reusability*, vol. 8769, pp. 287–303. Springer International Publishing, Berlin (2014)
18. Hametner, R., Winkler, D., Östreicher, T., Biffl, S., Zoitl, A.: The adaptation of test-driven software processes to industrial automation engineering. In: *IEEE International Conference on Industrial Informatics (INDIN)* (2010)
19. Hashtrudi Zad, S., Kwong, R.H., Wonham, W.M.: Fault diagnosis in discrete-event systems: framework and model reduction. *IEEE Trans. Autom. Control* **48**(7), 1199–1212 (2003)
20. Haubeck, C., Lamersdorf, W., Ladiges, J., Fay, A., Fuchs, J., Legat, C., Vogel-Heuser, B.: Interaction of model-driven engineering and signal-based online monitoring of production systems: towards requirement-aware evolution. In: *Conference of the IEEE Industrial Electronics Society (IECON)* (2014a)
21. Haubeck, C., Lamersdorf, W., Ladiges, J., Fay, A.: An active service-component architecture to enable self-awareness of evolving production systems. In: *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* (2014b)
22. Haubeck, C., Wior, I., Braubach, L., Pokahr, A., Ladiges, J., Fay, A., Lamersdorf, W.: Keeping pace with changes—towards supporting continuous improvements and extensive updates in production automation software. *Electron. Commun. EASST* **56**, 1–12 (2013)
23. Isermann, R.: *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer, Berlin, Heidelberg (2006)
24. ISO/IEC15909–1: *Software and System Engineering—High-Level Petri Nets—Part 1: Concepts, Definitions and Graphical Notation* (15909-1) (2004)

25. Ladiges, J., Fülber, A., Haubeck, C., Arroyo, E., Fay, A., Lamersdorf, W.: Learning material flow models for manufacturing plants from data traces. In: IEEE International Conference on Industrial Informatics (2015a)
26. Ladiges, J., Haubeck, C., Fay, A., Lamersdorf, W.: Operationalized definitions of non-functional requirements on automated production facilities to measure evolution effects with an automation system. In: IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (2013)
27. Ladiges, J., Haubeck, C., Fay, A., Lamersdorf, W.: Evolution management of production facilities by semi-automated requirement verification. *at-Automatisierungstechnik* **62**(11), 781–793 (2014)
28. Ladiges, J., Haubeck, C., Fay, A., Lamersdorf, W.: Learning behaviour models of discrete event production systems from observing input/output signals. In: IFAC/IEEE/IFIP/IFORS Symposium on Information Control Problems in Manufacturing (INCOM) (2015b)
29. Ladiges, J., Haubeck, C., Lity, S., Fay, A., Lamersdorf, W., Schäfer, I.: Supporting commissioning of production plants by model-based testing and model learning. In: International Symposium on Industrial Electronics (2015c)
30. Lange, J., Iwanitz, F., Burke, T.J.: OPC—From Data Access to Unified Architecture, 4th edn. VDE-Verlag, Berlin (2010)
31. Lefebvre, D., Leclercq, E.: Stochastic Petri net identification for the fault detection and isolation of discrete event systems. *IEEE Trans. Syst. Man Cybern. A Syst. Hum.* **41**(2), 213–225 (2011). doi:[10.1109/TSMCA.2010.2058102](https://doi.org/10.1109/TSMCA.2010.2058102)
32. Legat, C., Mund, J., Campetelli, A., Hackenberg, G., Folmer, J., Schütz, D., Broy, M., Vogel-Heuser, B.: Interface behavior modeling for automatic verification of industrial automation systems' functional conformance. *at-Automatisierungstechnik* (2014). doi:[10.1515/auto-2014-1126](https://doi.org/10.1515/auto-2014-1126)
33. Liebowitz, J.: Knowledge Management Handbook. CRC Press, Boca Raton (1999)
34. Lochau, M., Bürdek, J., Lity, S., Hagner, M., Legat, C., Goltz, U., Schür, A.: Applying Model-based Software Product Line Testing Approaches to the Automation Engineering Domain. *at-Automatisierungstechnik* **62**(11), 771–780 (2014)
35. Moore, K.E., Gupta, S.M.: Petri net models of flexible and automated manufacturing systems: a survey. *Int. J. Prod. Res.* **34**(11), 3001–3035 (1996). doi:[10.1080/00207549608905075](https://doi.org/10.1080/00207549608905075)
36. Niggemann, O., Frey, C.: Data-driven anomaly detection in cyber-physical production systems. *at-Automatisierungstechnik* **63**(10), 821–832 (2015)
37. Rogalski, S.: Flexibility Measurement in Production Systems: Handling Uncertainties in Industrial Production. Springer, Berlin (2011)
38. Roth, M., Lesage, J., Litz, L.: Black-box identification of discrete event systems with optimal partitioning of concurrent subsystems. In: American Control Conference (ACC) (2010)
39. Schneider, S., Litz, L., Lesage, J.: Determination of timed transitions in identified discrete-event models for fault detection. In: IEEE Annual Conference on Decision and Control (CDC) (2012)
40. Sethi, A., Sethi, S.: Flexibility in manufacturing: a survey. *Int. J. Flex. Manuf. Syst.* **2**(4), 289–328 (1990). doi:[10.1007/BF00186471](https://doi.org/10.1007/BF00186471)
41. Thramboulidis, K.C.: Using UML in control and automation: a model driven approach. In: IEEE International Conference on Industrial Informatics (INDIN) (2004)
42. Tompkins, J.A.: Facilities Planning, 4th edn. Wiley, Hoboken (2010)
43. Torens, C., Ebrecht, L., Lemmer, K.: Inverse model based testing—generating behavior models from abstract test cases. In: International Conference on Software Testing, Verification and Validation Workshops (2011)
44. VDI/VDE Society for Measurement and Automatic Control: VDI/VDE 3681-Classification and evaluation of description methods in automation and control technology (2005)
45. Vodencarevic, A., Kleine Büning, H., Niggemann, O., Maier, A.: Using behavior models for anomaly detection in hybrid systems. In: International Symposium on Information, Communication and Automation Technologies (2011)
46. Vogel-Heuser, B., Diedrich, C., Fay, A., Jeschke, S., Kowalewski, S., Wollschlaeger, M., Göhner, P.: Challenges for software engineering in automation. *JSEA* **07**(05), 440–451 (2014). doi:[10.4236/jsea.2014.75041](https://doi.org/10.4236/jsea.2014.75041)
47. Vogel-Heuser, B., Fay, A., Schäfer, I., Tichy, M.: Evolution of software in automated production systems: challenges and research directions. *J. Syst. Softw.* **110**, 54–84 (2015a)
48. Vogel-Heuser, B., Feldmann, S., Folmer, J., Kowal, M., Schäfer, I., Ladiges, J., Fay, A., Haubeck, C., Lamersdorf, W., Lity, S., Kehrer, T., Tichy, M., Getir, S., Ulbrich, M., Klebanov, V., Beckert, B.: Selected challenges of software evolution for automated production systems. In: IEEE International Conference on Industrial Informatics (2015b)
49. Vogel-Heuser, B., Schütz, D., Frank, T., Legat, C.: Model-driven engineering of Manufacturing Automation Software Projects—a SysML-based approach. *Mechatronics* **24**(7), 883–897 (2014b)
50. Vyatkin, V.: Software engineering in industrial automation: state-of-the-art review. *IEEE Trans. Ind. Inform.* **9**(3), 1234–1249 (2013). doi:[10.1109/TII.2013.2258165](https://doi.org/10.1109/TII.2013.2258165)
51. Wahab, M.I.M.: Measuring machine and product mix flexibilities of a manufacturing system. *Int. J. Prod. Res.* **43**(18), 3773–3786 (2005). doi:[10.1080/00207540500147091](https://doi.org/10.1080/00207540500147091)
52. Winkler, D., Biffl, S., Östreicher, T.: Test-driven automation-adopting test-first development to improve automation systems engineering processes. In: EuroSPI Conference (2009)
53. Witte, M.: System engineering, plant engineering and functional models. *Softwaretechnik-Trends* **32**(2), 94–95 (2012)
54. Zimmermann, A.: Stochastic Discrete Event Systems—Modeling, Evaluation, Applications. Springer, Berlin, Heidelberg, New York (2008)
55. ZVEI e.V.: Life-Cycle-Management for Automation Products and Systems: A Guideline by the System Aspects Working Group of the ZVEI Automation Division (2010)