Open access • Journal Article • DOI:10.1080/00207721.2011.649369

Fault detection and isolation in manufacturing systems with an identified discrete event model — Source link

Matthias Roth, Stefan Marco Schneider, Jean-Jacques Lesage, Lothar Litz

Institutions: École Normale Supérieure, Kaiserslautern University of Technology

Published on: 01 Oct 2012 - International Journal of Systems Science (Taylor & Francis Group)

Topics: Fault detection and isolation, Stuck-at fault and Fault (power engineering)

Fault Detection and Isolation in Manufacturing Systems with an Identi ed Discrete Event Model

Matthias Roth, Stefan Schneider, Jean-Jacques Lesage, Lothar Litz

Fault Detection and Isolation in Manufacturing Systems with an Identified Discrete Event Model

Matthias Roth[a] [b], Stefan Schneider [a] *, Jean-Jacques Lesage[b] and Lothar Litz[a]

[a] *Institute of Automatic Control, University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern, Germany*; [b] *LURPA - Ecole Normale Supérieur de Cachan, 61, Avenue du Président Wilson, 94235 Cachan Cedex, France*

(*December 2010*)

In this paper a generic method for fault detection and isolation (FDI) in manufacturing systems considered as discrete event systems (DES) is presented. The method uses an identified model of the closed loop of plant and controller built on the basis of observed fault free system behavior. An identification algorithm known from literature is used to determine the fault detection model in form of a non-deterministic automaton. New results of how to parameterize this algorithm are reported. To assess the fault detection capability of an identified automaton, probabilistic measures are proposed. For fault isolation, the concept of residuals adapted for DES is used by defining appropriate set operations representing generic fault symptoms. The method is applied to a case study system.

Keywords: Diagnosis of Discrete Event Systems, Identification of Discrete Event Systems

1. Introduction

Diagnosis of complex manufacturing systems is a challenging task placing high demands on development and application. For economic reasons industrial companies have a grown interest in increasing the availability of production systems. Fault detection and isolation concepts contribute significantly to this endeavor.

Automated manufacturing systems are usually controlled by a Programmable Logic Controller (PLC) and are often highly individual installations with many digital controller inputs and outputs (I/Os). The process is characterized by binary information exchange between plant and controller. Resulting dynamics are sequential occurrences of discrete events allowing to consider these systems as discrete event systems (DES).

In the last 10 to 15 years a lot of research on fault diagnosis of discrete event systems has been done. Most of the developed approaches are based on a diagnoser which has been introduced by Sampath et al. (1996). The diagnoser approach starts with building a model composed of the system components and of the controller program by including the fault-free, normal system behavior as well as the behavior in case of given faults. It contains unobservable and observable events. Faults are usually modeled as unobservable events. From this model a special observer called diagnoser is derived that allows detecting and diagnosing the occurrence of an anticipated fault by analyzing only observable events.

Another class of diagnostic approaches that works on the basis of event order and the timed behavior of a system is presented in Pandalai and Holloway (2000). This method does not

*Corresponding author. Email: sschneider@eit.uni-kl.de

work with a state estimation but with so called condition templates. It analyzes whether the considered system creates events in the right order or in given time delays. A fault is detected if there are missing or wrong reactions in the process.

Sayed-Mouchaweh et al. (2008) presented a combination of these two kinds of approaches that is adapted for fault diagnosis of manufacturing systems. Instead of one central diagnoser, a set of decentralized diagnosers is developed that delivers diagnosis information according to special rules. The timed behavior of the considered system is monitored with condition templates in each state of the diagnoser. The method needs detailed models of the system components as well as the determination of the timed system behavior.

The classical diagnoser approach has been adapted by several authors in the past. Hashtrudi Zad et al. (2005) e.g. extended the approach by considering the timed behavior by introducing timing events. With a special time event it is possible to determine the period between two expected events. Like in the original approach a diagnoser can be constructed using the component models.

As a common drawback of these approaches a high degree of system knowledge is needed. The manual model building or determination of event orders is a laborious task that can make it impossible from an economical point of view to install an FDI system. Especially for already existing and operating manufacturing systems with many I/Os it can be hard to obtain the necessary information such as models for non standard components or a formal description of the controller program. The main contributions of the new attempt proposed in this paper are automatic model building and using models without explicitly included faulty behavior. Event sequences are gathered from the system by observing the nominal, fault-free system behavior. In contrast to the approaches presented in literature the model building process is performed utilizing only very little knowledge of the system. This enables building FDI systems for complex manufacturing facilities with reasonable effort. Time aspects of the DES are not considered in this paper. A possible way of how to use time information in system models for fault detection and isolation purposes is proposed in Schneider et al. (2011).

The presented FDI attempt is contrasted in detail with the diagnoser and condition templates approach in Danancher et al. (2011). Comparison criterions are model size, ease of model building, diagnosability property and false alerts. By means of a case study the FDI methods are evaluated with respect to fault detection and isolation capabilities.

Besides automata, Petri Nets are a suitable framework for system modeling. An overview of fault diagnosis using Petri Nets is given in Fanti and Seatzu (2008). Most recent developments are related to integer linear programming (ILP) techniques to solve the diagnosis task. In Dotoli et al. (2009) e.g. such an approach is proposed. Unobservable transitions in a Petri Net are used to model faults. In order to decide whether the system behavior is consistent with the modeled normal behavior or if it must be explained by modeled (unobservable) fault events, integer linear programming problems are solved. The major drawbacks of this method are first: to ensure accurate fault detection the initial state of the system must be known and second: because of the high calculation efforts necessary to solve the integer linear programming problems the method is not applicable for large scale systems.

A detailed comparison and evaluation of identification approaches for DES to the one applied in this work is subject of Estrada-Vargas et al. (2010). The two other described methods are based on petri net identification. One is related to the introduced linear integer programming approach, the other one is known as progressive identification (Meda-Campana 2002). Different characteristics of the methods are figured out to highlight the appropriate methods for automatic building of DES models.

The advantage of Petri Nets is the possibility to visualize concurrent and causal system behavior. Using identified Petri Nets may allow to get a deeper insight into an unknown system structure by analyzing the resulting graph. The main interest of the presented FDI approach is to compare the observed and modeled output behavior hence we prefer using a compact au-

tomaton model. It is not out intention to get a deeper system insight based on the identified model.

Since model building is the main obstacle for the use of model-based diagnosis techniques an approach based on an identified model is presented in this paper. Only very little physical knowledge of the system is necessary. In the following parts of the paper it will be explained how an appropriate model of an automated system can be identified. Since the model is identified on the basis of observed fault-free system behavior, the model only represents the fault free behavior. Once the model has been identified, it will be explained how it can be used for fault detection and fault isolation. *The fault detection policy is to consider each trajectory which cannot be reproduced by the model as a fault.* A probabilistic measure is introduced which helps assessing the fault detection capability of an identified model using this policy.

The paper is intended to give an overview of the identification based FDI method. In some parts of the paper the reader will thus be referred to the according literature to get more details. Note that although the outline of the diagnosis follows model identification, it is also possible to use manually-built models.

2. Overview of the proposed method

Systems considered in this paper are DES consisting of a closed-loop of plant and controller. The controller has a certain number of binary outputs connected with actuators in the plant. Sensors of the plant are connected to binary controller inputs. Possible faults in this scenario are broken actuators and sensors as well as damaged plant hardware. In our approach, the only information used to perform the identification and diagnosis task are the signals exchanged between controller and plant. Our aim is to detect if a sensor or an actuator fails or if some plant hardware gets damaged which leads to an abnormal behavior. Recognizing such an abnormal behavior will be referred to as fault detection. In order to allow systematic repair actions, it is necessary to isolate faulty components once a fault has been detected. The aim of fault isolation in our context is to determine controller I/Os (inputs and outputs) which are related to a faulty component.

Figure 1 shows the principle of the diagnosis approach used in this work. The considered closed-loop DES exhibits a system output when performing its operation. The system output is defined by the signals exchanged between controller and plant. The system output is given to an evaluator containing the identified fault-free system model. The evaluator estimates the current state of the system with the identified model. Once the current state is estimated, the observed system output and the model output are compared. Any deviation leads to fault detection and appropriate fault isolation procedures are started. If there is no difference between observed and expected behavior, the system is supposed to work under fault-free conditions and no fault is detected.

The general idea of the approach is that in each system state only certain system outputs are allowed as valid following behavior. If the system model cannot reproduce the observed system output starting in the estimated state, a fault must have occurred. This necessitates on the one hand that the model can reproduce each fault-free following behavior from a given state to avoid false alerts. On the other hand, the model should not be able to produce other following behaviors than the fault-free ones starting in a given state in order to guarantee that faulty behavior cannot be erroneously reproduced which would lead to non-detectable faults.

To apply this diagnosis principle, it is necessary to have a model of the fault-free system behavior. Figure 2 sketches how this model is obtained. The signals exchanged between controller and plant of the closed-loop DES are captured during fault-free system evolutions and stored in a data base. At this point it is necessary to decide that a given system evolution can be considered as fault-free. This decision can for example be taken based on an analysis of the

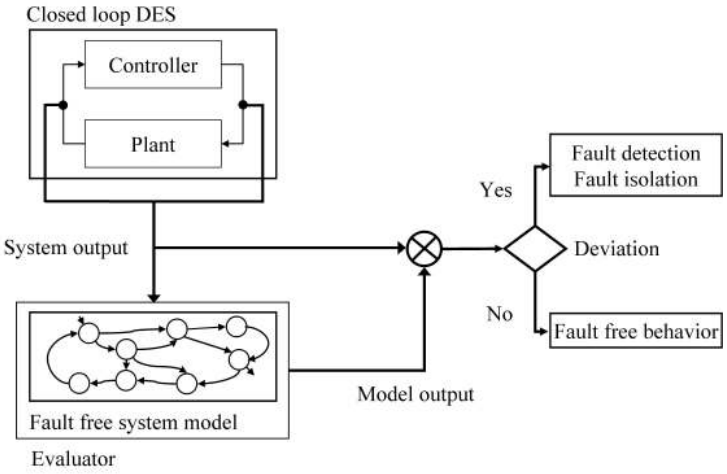

Figure 1. Model-based FDI principle

quality of the produced items. Although we use the term 'fault-free' behavior in this work, it rather reflects the *acceptable* one. This makes it possible that small disturbances are not directly considered as faulty behavior and can thus be integrated in the model. Since the model contains these small disturbances, it is possible to avoid a huge number of unimportant fault detections which increases the use of the diagnosis system. It can be seen that the data base consists of sequences of signal vectors. This data structure will be defined in the next section. Based on the data base, an identification algorithm is applied delivering a model which is able to reproduce the observed fault-free system behavior.

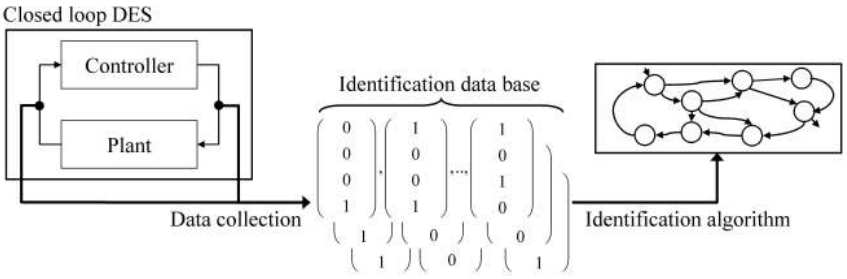

Figure 2. Principle of model identification

3. Data collection and data structure

As explained in the former section, it is necessary to collect the signals exchanged between controller and plant for identification and for diagnosis purposes. A data collection approach that can easily be implemented using standard controller hardware like programmable logic controllers (PLC) is to collect the signals after they have been captured by the controller (Roth et al. 2010). Figure 3 shows the functional principle of a PLC which is a widely used class of controllers in industry. The controller cyclically performs the steps 'input reading' where it reads the signals from the sensors, 'program execution' to determine new output values for the actuators, and 'output writing' where the newly determined commands are sent to the plant actuators. Modern PLCs are equipped with a communication processor which makes it possible to send the values of the input and output signals to a standard PC where they can be stored in a data base.

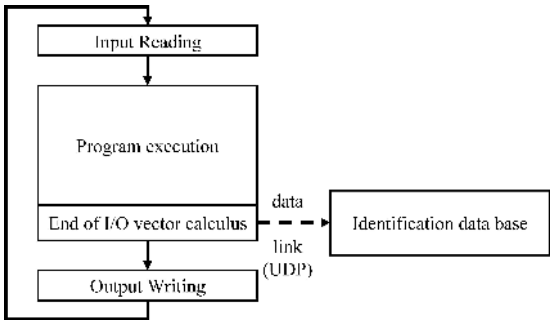

Figure 3. Collection of signals exchanged between controller and plant

Once the data collection has been implemented, the collected data has to be structured. The idea of the data structure used in this work is to arrange the collected signals in an I/O vector.

Definition 3.1: Controller I/O vector: Given r different controller inputs I_1, ..., I_r and s different controller outputs O_1, ..., O_s, the controller I/O vector $u = (IO_1, ..., IO_m)$ with $m = r + s$ is given by $IO_i = I_i \; \forall \; i = 1, .., r$ and $IO_{r+i} = O_i \; \forall \; i = 1, .., s$. The dimension (number of controller I/Os) of the vector is denoted $m = |u|$. The controller I/Os can have the values 1 or 0.

The system is exhibiting a series of controller I/O vectors when performing an evolution. The output is defined as follows.

Definition 3.2: System output: The j-th output in the h-th of p system evolutions is defined as $u_h(j)$.

The identification of the model is based on sequences of observed system outputs (I/O vector sequences). The output sequence of I/O vectors that is exhibited during a given system evolution is built by I/O vectors in the order of their appearance.

Definition 3.3: Observed sequence: If during the h-th system evolution l_h outputs u_h have been observed, the sequence is denoted as $\sigma_h = (u_h(1), u_h(2), \ldots, u_h(l_h))$. The set of all observations is denoted as $\Sigma = \{\sigma_1, \ldots, \sigma_p\}$.

Hence l_h denotes the length of the h-th I/O vector sequence. We assume that for two successive I/O vectors $u(t) \neq u(t-1)$ holds. This assumption makes sure that an I/O vector is only considered as a new one if at least one I/O changed its value. In order to translate the observed I/O vector sequences into a model, a formal definition of the observed system behavior is necessary. We define the set of observed words with length q observed during p different system evolutions.

Definition 3.4: Observed word set and language: The words of length q observed during p different system evolutions are denoted as

$$W_{Obs}^{q} = \bigcup_{\sigma_h \in \Sigma} \left(\bigcup_{j=1}^{|\sigma_h|-q+1} (u_h(j), u_h(j+1), \ldots, u_h(j+q-1)) \right).$$

The length of the h-th observed sequence is denoted as $|\sigma_h|$. With the observed word set the observed system language of length n is defined as follows.

$$L_{Obs}^{n} = \bigcup_{i=1}^{n} W_{Obs}^{i}.$$

The observed language of length n consists of the I/O vector sequences up to length n that have been observed to get the necessary data base for identification.

4. Model identification

4.1. *Model class*

The aim of identification is to get a model which is able to reproduce the language of the considered system. Hence, it should reflect the characteristics of a closed-loop DES. The closed-loop DES is an autonomous event generator that exhibits a system output. A well programmed controller can be considered as deterministic, whereas the physical plant must generally be considered as non-deterministic. Hence the coupled system of plant and controller must be considered as non-deterministic. An automaton that reflects these properties is the **N**on-**D**eterministic **A**utonomous **A**utomaton with **O**utput (Klein 2005) that is defined as follows.

Definition 4.1: Non-deterministic autonomous automaton with output (NDAAO): $NDAAO = (X, \Omega, f, \lambda, x_0)$ with X finite set of states, Ω output alphabet, $f : X \to 2^X$ non-deterministic transition function, $\lambda : X \to \Omega$ output function and x_0 the initial state.

The dynamics of the automaton are defined as follows.

Definition 4.2: Next-state rule NDAAO: A NDAAO state $x(j)$ can be the successor of the last current state $x(j-1)$ if and only if $x(j) \in f(x(j-1))$. If there are several next state candidates ($|f(x(j-1))| > 1$) the new actual state $x(j)$ is chosen non-deterministically from the set $f(x(j-1))$.

With this definition it is possible to define the language of the NDAAO according to the language of the considered closed-loop DES.

Definition 4.3: Word set and language of the NDAAO: The set of words of length n generated from a state $x(i)$ is defined as:

$$W_{x(i)}^{n=1} = \{w \in \Omega^1 \mid w = \lambda(x(i))\}$$

and

$$W_{x(i)}^{n>1} = \{w \in \Omega^n \mid \big(w = (\lambda(x(i)), \lambda(x(i+1)), \ldots, \lambda(x(i+n-1))) \wedge x(j+1) \in f(x(j)) \ \forall i \leq j < i+n-1\big)\}$$

The language generated by the NDAAO is given by

$$L_{Ident}^n = \bigcup_{i=1}^{n} \bigcup_{x \in X} W_x^i$$

If the output alphabet Ω consists of I/O vectors, it is possible to reproduce the observed language by performing state trajectories in the automaton.

4.2. *Motivation for a history parameter*

Before the identification algorithm is sketched in the next section, it is shown why an identification parameter is necessary to guarantee a sufficient model accuracy if the model is used for online diagnosis purposes. The idea of the parameter is to allow distinguishing non-equivalent physical states of a closed-loop DES leading to identical outputs.

In a closed-loop DES it is possible that different states lead to the same output (I/O vector) but have different fault-free following behaviors. As an example, Figure 4 is considered. The figure shows a conveyor as typical part of a manufacturing system. The conveyor and its controller can be treated like a closed-loop DES. The conveyor has three position sensors P1, P2 and P3. The according controller-inputs are the first three elements of the I/O vector. If the position sensors detect a parcel, they return the value 1, else they return 0. The conveyor can be started by setting an actuator to 1. Figure 4 shows the parcel in two different positions (position I and position II). It can be seen that the two positions lead to the same system output (C) but have a different fault-free following behavior: In position I, P2 is expected to change its value (leading to vector D) and in position II, P3 is expected to change its value (leading to vector E). The two underlying closed-loop DES states are thus not equivalent although they lead to the same output vector: According to Cassandras and Lafortune (2006) and Klein (2005), two states are equivalent if they have the same output *and* if the language starting in them is equal.

An accurate model should be able to distinguish situation I and II although they are represented by the same I/O vector. An automaton of the type NDAAO that can distinguish between the two non-equivalent system states is given in the right under the conveyor in Figure 4. Position I corresponds to state x_2 and position II belongs to state x_4. If the two non-equivalent closed-loop DES states are *not* represented by two different states in an automaton like in the left part of Figure 4, the following situation can arise: A fault lets the system perform the sequence BCE which is not part of the fault-free behavior. BCE is a fault symptom since the parcel must pass sensor P2 before it reaches P3 which is represented by I/O vector E. The automaton on the left has a state trajectory which leads to the generated word BCE (x_1, x_2 and x_4). The reason is that x_2 can produce the following behavior of both underlying non-equivalent closed-loop DES states. This makes the detection of the considered fault impossible with the automaton in the left of Figure 4. If a model in form of an automaton is built manually, an intuitive way to achieve high accuracy is to represent only *equivalent* DES states with one automaton state. Two automaton states are intuitively connected if the represented DES states can occur successively.

If a model is built by identification, the decision if two identical system outputs belong to two different closed-loop DES states must be taken based on an analysis of the observed system data. In the example in Figure 4 it can be seen that using the preceding I/O vector of position I and II allows distinguishing the two underlying closed-loop DES states: If vector C occurs due to position I, it must be the successor of vector B. If C is caused by position II, it must follow vector D.

Following this considerations, the idea of the following identification algorithm can be summarized as follows: An identification parameter k defines the length of I/O vector sequences which are analyzed to create new states. For each new sequence of length k an own state is created. Two states are only connected if they share the same memory of length $k-1$. If k is chosen such that it is possible to distinguish two non-equivalent closed-loop DES states with the same output by their preceding sequences of length k, the identification algorithm will not represent them with the same automaton state. In this case it will be made sure that each automaton state can only produce the fault-free following behavior of *one* closed-loop DES state. Any other possibly faulty following behavior cannot be produced. This minimizes the number of non-detectable faults during online fault diagnosis. In the example of Figure 4 it is sufficient to chose $k = 2$ since the non-equivalent closed-loop DES states with the same output are distinguishable by considering sequences of length two.

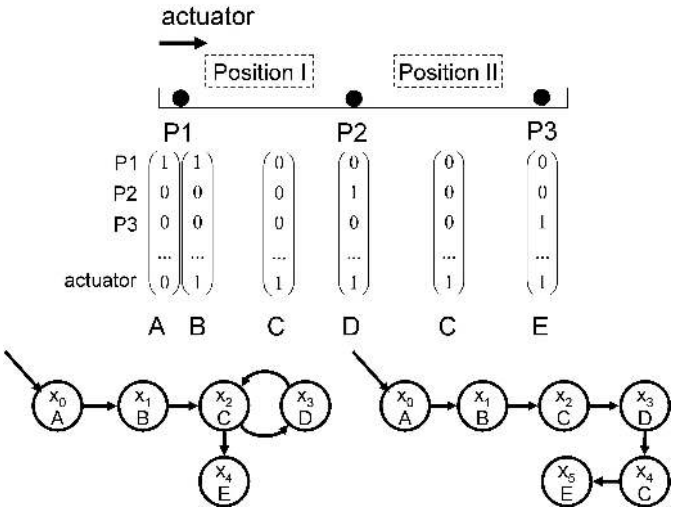

Figure 4. Meaning of k

4.3. *Identification algorithm*

As explained in Section 2, the collected data base has to be translated into a model using an identification algorithm. For this purpose, in Klein (2005) and Klein et al. (2005) an appropriate algorithm is proposed. The algorithm uses an identification parameter k which can be determined based on the considerations of the previous section. To get a basic understanding of the identification procedure, the algorithm will be presented in an informal way using an example. For a formal description and the proofs of the properties mentioned in this section, the reader is referred to the cited literature.

It is assumed that the following three sequences have been observed during fault-free system behavior and build the identification data base: $\sigma_1 = (A, B, C, D, E, A)$, $\sigma_2 = (A, D, B, C, D, A, C, A)$ and $\sigma_3 = (A, D, B, C, F, D, E, B)$. The letters represent different I/O vectors and are used to facilitate reading the example. For the example, the identification parameter k is chosen to $k = 2$.

As explained in Section 4.2, the idea of the algorithm is to create a new state for each system output if it differs from an already treated output in its preceding sequence of length k. Since the first I/O vectors in an observed sequence do not have preceding sequences, they are duplicated $k-1$ times to allow distinguishing them from non-initial I/O vectors with a different preceding sequence.

$$\sigma_h^k(i) = \begin{cases} \sigma_h(1) \text{ for } 1 \leq i \leq k \\ \sigma_h(i-k+1) \text{ for } k < i \leq k + |\sigma_h| - 1 \end{cases} \tag{1}$$

The length of the h-th observed sequence is denoted as $|\sigma_h|$. Equation (1) is applied to each sequence $\sigma_h \in \Sigma$ resulting in $\Sigma^k = \{\sigma_1^k, \ldots, \sigma_p^k\}$.

In the example, Equation (1) is applied to the three sequences to duplicate the first output symbol $k-1$ times. The result is

$$\sigma_1^{k=2} = (A, A, B, C, D, E, A)$$
$$\sigma_2^{k=2} = (A, A, D, B, C, D, A, C, A)$$
$$\sigma_3^{k=2} = (A, A, D, B, C, F, D, E, B)$$

On this basis, the modified word sets of length k and $k+1$ are determined according to Definition 3.4:

$$W^k_{Obs,\Sigma^k} = \bigcup_{\sigma_h^k \in \Sigma^k} \left(\bigcup_{i=1}^{|\sigma_h^k|-k+1} (u_h(i), u_h(i+1), \ldots, u_h(i+k-1)) \right) \tag{2}$$

$$W^{k+1}_{Obs,\Sigma^k} = \bigcup_{\sigma_h^k \in \Sigma^k} \left(\bigcup_{i=1}^{|\sigma_h^k|-k} (u_h(i), u_h(i+1), \ldots, u_h(i+k)) \right) \tag{3}$$

In the example, W^k_{Obs,Σ^k} and W^{k+1}_{Obs,Σ^k} follow from Equation (2) and (3):

$$W^2_{Obs,\Sigma^2} = \{AA, AB, BC, CD, DE, EA, EB, AD, DB, DA, AC, CA, CF, FD\}$$

and

$$W^3_{Obs,\Sigma^2} = \{AAB, ABC, BCD, CDE, DEA, DEB, AAD, ADB,$$
$$DBC, CDA, DAC, ACA, BCF, CFD, FDE\}$$

It can be seen that in W^2_{Obs,Σ^2} the sequences AA and DA allow distinguishing two different closed-loop DES states leading to the output A.

The data basis for the identification algorithm is given by W^k_{Obs,Σ^k} and W^{k+1}_{Obs,Σ^k}.

The algorithm consists of the following steps.

1. The state space of the NDAAO is built: For each word $w^k \in W^k_{Obs,\Sigma^k}$ a state x is built and its output function is associated with this word. After this step, the automaton consists of isolated, non-connected states. In the example, for each word in W^2_{Obs,Σ^2}, a state is created, see the 14 states in Figure 5.

2. Creation of the transition function: In this step, the states are connected. The transition function of the NDAAO is built on the basis of W^{k+1}_{Obs,Σ^k}. For each word w^{k+1} in this set the first k-long subsequence and the last k-long subsequence are determined. For each substring there exists exactly one state which has this string as output. The two according states x and x' are selected and get connected such that the state representing the first substring gets the state of the second substring added to its successor states. In the example, state x_0 has been connected to states x_1 and x_6 because of the words $w^3 = AAB$ and $w^3 = AAD$. $w^3 = AAB$ was divided in the two substrings AA and AB. Hence, states x_0 with $\lambda(x_0) = AA$ and x_1 with $\lambda(x_1) = AB$ have been connected.

3. Determination of the initial state: The state representing the sequence with the $k-1$-times duplicated first output symbols becomes the initial state. In the example, this state is x_0.

4. Redefinition of the output function: The output function of the states is reassigned with the last 'letter' of the output word. The result of this step can also be seen in Figure 5: the words in the states are replaced by the last letters which are written in bold and italic.

5. Reduction of the state space: To reduce the state space, equivalent states are merged. According to Cassandras and Lafortune (2006), two automaton states x_1, x_2 are equivalent if the language starting in them are equal. In Klein (2005) it is shown that this condition is fulfilled for two NDAAO-states if they are associated with the same output, i.e. $\lambda(x_1) = \lambda(x_2)$ and if they have the same set of following states, i.e. $f(x_1) = f(x_2)$. In the example, states x_5 and x_{10} as well as x_1 and x_7 are merged.

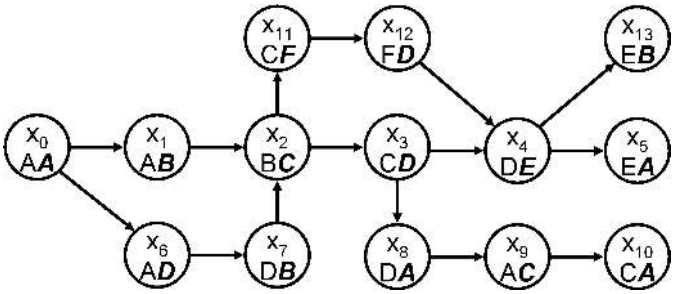

Figure 5. Identified NDAAO after the second step of the identification algorithm

The result of the algorithm after merging of equivalent states can be seen in Figure 6. The NDAAO identified with $k = 2$ allows distinguishing between two non-equivalent closed-loop DES states leading to the same output if their preceding sequences of length k are distinguishable. In the example, this allows distinguishing x_3 and x_{12} although they have the same output. The advantage for fault diagnosis purposes is obvious: If the closed-loop DES is in the state represented by x_{12} and a fault leads to the following output A, this fault can be detected if the current state is correctly estimated (the automaton cannot create A starting in x_{12}). If each closed-loop DES state leading to D were represented with the same NDAAO state, this state would allow each possible following behavior of the represented closed-loop DES states. This would make it impossible to detect a fault upon the observation of the I/O vector A in the described scenario.

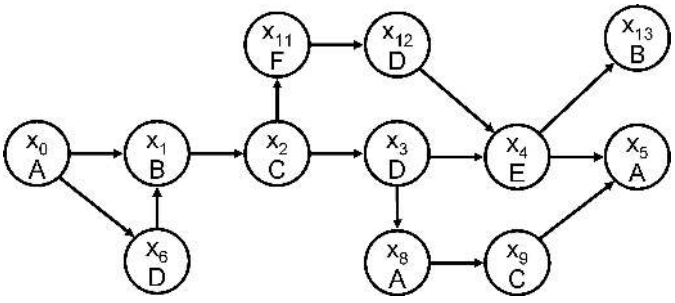

Figure 6. The identified NDAAO after merging of equivalent states

An important characteristic of the identified NDAAO is shown in Klein (2005): An automaton identified with a given parameter k is $k+1$-complete in the sense of Moor et al. (1998), $L_{Ident}^{k+1} = L_{Obs}^{k+1}$. This means the automaton reproduces exactly the observed words of length $k+1$. If the identification data base has been collected during fault-free system behavior only, this means that we can be sure that the automaton does not produce faulty words of length $k+1$.

A second important property of the identified model is shown in Roth et al. (2010). If the observed language of length $k+1$ can be considered as completely observed (each word of length

$k+1$ of the original system language has been seen during the data collection phase), a NDAAO identified with a given parameter k simulates the complete original system language. In this case, the model is able to reproduce each fault-free word of the original system language. Hence, it can be expected that there will be no false alerts using the model for online diagnosis. In Section 6 it will be shown how it can be decided that the language of length $k+1$ can be considered as completely observed.

5. Fault Detection and Isolation

5.1. *State estimation and fault detection*

In the former section it has been explained that an accurate state estimation is necessary to detect faults with the identified model. With Algorithm **1** such a state estimation is possible using a NDAAO. The main idea of this algorithm is to determine a current state estimation following the I/O vectors observed from the system. $\lambda(x)$ determines the output of a NDAAO state according to Definition 4.1. With $\widetilde{X}_{t-1}$ and $\widetilde{X}_t$ we denote the state estimations after the occurrence of the $t-1$-th and t-th I/O vector respectively. Before the algorithm is started, both sets are empty.

The algorithm checks if the former state estimation $\widetilde{X}_{t-1}$ contains at least one state. If the observation is being initialized or after a fault has been detected, this set is empty. In case of an empty estimation $|\widetilde{X}_{t-1}| = 0$, the algorithm determines each NDAAO state with the observed I/O vector as output as possible current state (see line 4 of Algorithm **1**) and adds it to the current state estimation $\widetilde{X}_t$. If the former state estimation $\widetilde{X}_{t-1}$ was not empty (line 2), the algorithm analyzes each direct following state x' of states in $\widetilde{X}_{t-1}$ ($x' \in f(x)$ with $x \in \widetilde{X}_{t-1}$). Each x' with the newly observed I/O vector as output ($\lambda(x') = u(t)$) is added to the current state estimation $\widetilde{X}_t$. At the end of the algorithm, the former state estimation $\widetilde{X}_{t-1}$ for the next run of the algorithm is prepared by copying $\widetilde{X}_t$ to it.

Algorithm 1 Evaluator algorithm

Require: New observed I/O vector $u(t)$ and former state estimation $\widetilde{X}_{t-1}$
1: **if** $|\widetilde{X}_{t-1}| > 0$: **then**
2: $\quad \widetilde{X}_t := \{x' \in X | \big(\exists x \in \widetilde{X}_{t-1} \wedge x' \in f(x) \wedge \lambda(x') = u(t)\big)\}$
3: **else**
4: $\quad \widetilde{X}_t := \{x \in X | \lambda(x) = u(t)\}$
5: **end if**
$\widetilde{X}_{t-1} := \widetilde{X}_t$
6: **return** $\widetilde{X}_t$

Based on the result of the state estimation, a fault detection policy can be defined: A fault is detected if the current state estimation is empty. In this case, the algorithm was not able to reproduce the currently observed I/O vector. Hence the vector has to be considered as a fault symptom.

A special challenge for diagnosis system dedicated to industrial systems is the handling of false alerts. In Izadi et al. (2009) the results of an study considering alarm management in diagnosis systems of large plants are outlined. Between 36 and 48 average alarms per hour are typical values for alarm systems in industrial environment. Hence, a model based diagnosis method should be able to cope with false alerts. For automaton-based approaches this means that the state estimation algorithm has to be efficient if the model is reinitialized after a false alert. In Roth (2010) it is shown that using a model identified with parameter k, the state

estimation algorithm needs at most k fault-free I/O-vectors to determine a unique automaton state as current state. Hence, if due to a false alert the state estimation algorithm does not deliver a current NDAAO state, it takes at most k I/O vectors to reinitialize the model which allows restarting diagnosis.

5.2. *Fault isolation*

Once a fault has been detected, the next step is to isolate components which could be the reason for the malfunctioning. In the considered class of closed-loop DES, components are directly related to controller I/Os (e.g. sensors to controller inputs). The fault isolation strategy is to determine controller I/Os which have shown an abnormal behavior. With this knowledge it is possible to find the related components for a deeper analysis by the maintenance crew. The challenge is to determine a *small* set of possibly faulty controller I/Os such that the maintenance crew has only to check a limited number related components. Hence, we have to be able to determine the behavior of single I/Os based on the observed and modeled I/O vectors. If the system produces a new I/O vector, it is the result of at least one I/O changing its value. If a binary I/O changes its value a rising or falling edge can be observed (see Figure 7):

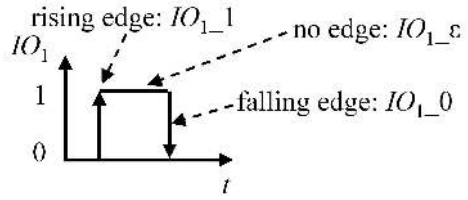

Figure 7. Rising and falling edge of a controller I/O

Definition 5.1: Edges: For each controller I/O IO_i three edges exist: IO_i_0 to indicate a change from 1 to 0 (falling edge), IO_i_1 to indicate a change from 0 to 1 (rising edge) and IO_i_ε to indicate no change in value:

$$E = \{IO_i\_0, IO_i\_1, IO_i\_\varepsilon \quad \forall 1 \leq i \leq m\}$$

with m denoting the number of controller I/Os in the system.

In order to determine the edges that appear if two arbitrary I/O vectors are considered, an edge function is defined:

Definition 5.2: Edge function: Let $IO_i(j)$, $IO_i(k)$ be the i-th controller I/O in the j-th and k-th I/O vector.

$$Edge(IO_i(j), IO_i(k)) = \begin{cases} IO_i\_1 & \text{if } IO_i(j) = 0 \quad \text{and} \quad IO_i(k) = 1 \\ IO_i\_0 & \text{if } IO_i(j) = 1 \quad \text{and} \quad IO_i(k) = 0 \\ IO_i\_\varepsilon & \text{if } IO_i(j) = IO_i(k) \end{cases}$$

delivers the resulting edge of the i-th I/O from the comparison of its values from the controller I/O vectors $u(j)$ and $u(k)$.

Instead of $Edge(IO_i(j), IO_i(k))$ we also write $Edge(u(j)[i], u(k)[i])$, i.e. we address the i-th controller I/O of vector u with $u[i]$. Since more than one I/O can change its value when new I/O vectors are produced, we define the evolution set that summarizes the edges 'between' two I/O vectors:

Definition 5.3: Evolution set:

$$ES(u(j), u(k)) = \bigcup_{i=1}^{m} \{Edge(u(j)[i], u(k)[i]) \in E \backslash IO_i\_\varepsilon\}$$

determines the set of rising and falling edges between two I/O vectors $u(j)$ and $u(k)$.

Figure 8 shows an example for the evolution set resulting from the comparison of two I/O vectors.

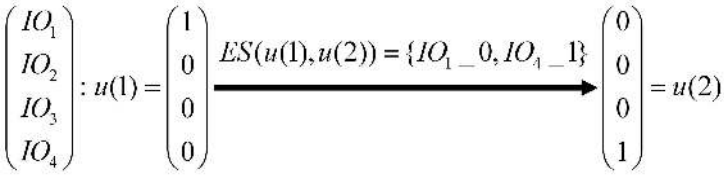

Figure 8. Example for the evolution set

Based on the evolution set it is now possible to analyze the behavior of single controller I/Os. In the following, two residuals will be introduced that formalize generic fault symptoms based on controller I/O behavior. The idea of the residuals follows the definition of Isermann and Balle (1997) where they are defined as fault indicators based on a deviation between *measurements* and *model-based computation*. In order to calculate the residuals, the state estimation before a fault was detected is supposed to be unambiguous $|\widetilde{X}_{t-1}| = 1$. Hence, only one NDAAO state was formerly considered as possible current fault-free state. If this condition does not hold, the residuals can not be calculated. If the state estimation algorithm uses a model which represents (almost) the complete fault-free system behavior, this is not very restrictive. In this case, there will be few or no false alerts which makes the state estimation algorithm delivering a unique state estimate most of the time. Due to the overview character of this paper, the residuals are not treated in every detail. For a more detailed description of the method, the reader is referred to Roth et al. (2009).

The first residual has the aim to isolate faults leading to an observed behavior that was unexpected in the given context. The current context is defined by the last estimated actual state $\widetilde{x}$ in the automaton. The first residual is

$$Res1(\widetilde{x}, u(t)) = ES(\lambda(\widetilde{x}), u(t)) \backslash \bigcup_{\forall x' \in f(\widetilde{x})} ES(\lambda(\widetilde{x}), \lambda(x')) \tag{4}$$

With $ES(\lambda(\widetilde{x}), u(t))$ the rising and falling edges are determined that are observed when comparing the I/O vector of the last estimated current state ($\lambda(\widetilde{x})$ with λ denoting the NDAAO state output function from Definition 4.1) and the I/O vector that led to fault detection. This set represents what actually happened when the fault was detected and refers to the notion of *measurements* in the residual definition. The union of the sets of rising and falling edges when the last estimated current state and each of its direct successor states are considered is represented by $\bigcup_{\forall x' \in f(\widetilde{x})} ES(\lambda(\widetilde{x}), \lambda(x'))$. It represents the expected behavior. This refers to the *model-based computation* in the residual definition. The set difference of the observed ($ES(\lambda(\widetilde{x}), u(t))$) and the expected ($\bigcup_{\forall x' \in f(\widetilde{x})} ES(\lambda(\widetilde{x}), \lambda(x'))$) behavior is built in the residual equation and thus represents the *unexpected* behavior.

Figure 9 shows an example. Since for the calculation of the residuals only the I/O edges are of interest, the state outputs ($\lambda(x)$, I/O vectors) are not shown. In the example it is assumed that the currently estimated state of the NDAAO is x_1 when an I/O vector occurs leading to a falling

edge of IO_3: IO_3_0. Since this cannot be reproduced by taking any of the leaving transitions of x_1, a fault is detected and fault isolation starts. The first residual is calculated as follows:

$$Res1(\widetilde{x}, u(t)) = \{IO_3\_0\} \backslash (\{IO_1\_0, IO_2\_1\} \cup \{IO_1\_0\}) = \{IO_3\_0\}$$

Hence, IO_3 is determined as a candidate for a fault leading to unobserved behavior.

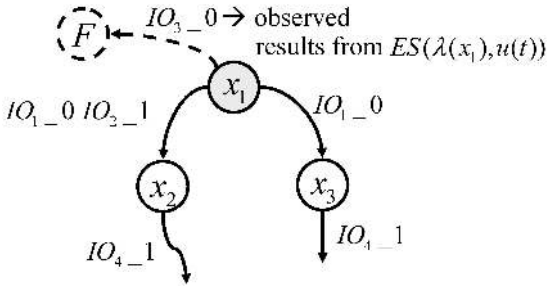

Figure 9. Example for the residuals

In contrast to an observed but unexpected behavior it is also possible that a fault can be isolated by determining a missed event. Set operations that help to isolate an expected but unobserved behavior are given by the third residual (the second residual can be found in Roth et al. (2009)).

$$Res3(\widetilde{x}, u(t)) = \bigcap_{\forall x' \in f(\widetilde{x})} ES(\lambda(\widetilde{x}), \lambda(x')) \backslash ES(\lambda(\widetilde{x}), u(t)) \tag{5}$$

$Res3$ is the set difference of the I/O edges that are expected no matter which following state is taken ($\bigcap_{\forall x' \in f(\widetilde{x})} ES(\lambda(\widetilde{x}), \lambda(x'))$) and the I/O edges that have been observed ($ES(\lambda(\widetilde{x}), u(t))$). Each rising or falling edge that must occur when the estimated actual state is left but has not been observed is part of $Res3$. The expected behavior is represented by the intersection of each possible following behavior.

As an example, $Res3$ is also calculated for the example in Figure 9:

$$Res3(\widetilde{x}, u(t)) = (\{IO_1\_0, IO_2\_1\} \cap \{IO_1\_0\}) \backslash \{IO_3\_0\} = \{IO_1\_0\}$$

It can be concluded that IO_1 could be affected by a fault leading to a missed behavior.

In general, all residuals have to be calculated if a fault is detected. Especially if large systems with many controller I/Os are considered, it is often possible to determine a small set of controller I/Os which are related to 'suspicious' components using the residuals. With the information if the according I/O was reported due to a missed or due to an unexpected behavior, the maintenance crew can start analyzing the according components.

5.3. *Fault detection capability of the model*

In many diagnosis approaches for DES, the notion of *diagnosability* plays an important role. In the sense of Sampath et al. (1996) this notion means the capability of a given diagnoser to diagnose a fault modeled as an unobservable event. If the property holds and if the modeled

fault leads to the behavior anticipated in the diagnosis model, it is guaranteed that the fault can eventually be found since the observed faulty behavior will eventually be distinguishable from the fault-free behavior.

In the presented approach the diagnosis model does not contain the behavior induced by faults since it has been identified based on observed fault-free system behavior. Hence, the classical diagnosability property is not helpful to assess the fault detection quality of an identified model. Since a measure of the fault detection capability is crucial to assess if the identified model is useful for online diagnosis purposes, an alternative approach is presented in this section.

The idea of the following measure is to calculate the probability that the model accepts an I/O vector which is induced by a fault. If this probability is sufficiently low, the model is appropriate for online diagnosis since it is not very likely that the model accepts faulty behavior.

First, we define three probabilistic events:

- F: Fault leading to an altered behavior of at least one controller I/O in the next I/O vector
- E_i: An I/O vector resulting from i edges occurs
- A: An I/O vector does not lead to fault detection in the evaluator algorithm (it can be reproduced by one of the following states of the current state)

With this definition of F, faults are only considered if they are potentially detectable by analyzing the next occurring I/O vector. Sooner or later most faults fall in this category even if they are not immediately detectable upon their appearance: Either they lead to a change in value of an I/O which is not expected or they prevent an I/O from changing its value although this is expected. Using these basic events, the probability of accepting an I/O vector induced by a fault is given by

$$P(A|F) = \sum_{i=1}^{m} P(A \cap E_i|F) \tag{6}$$

with m denoting the number of controller I/Os. The probability of $A \cap E_i$ under the condition F is $P(A \cap E_i|F)$ which means the probability of accepting an I/O vector resulting from i edges under the condition that there is a fault leading to an altered behavior of at least one controller I/O in the considered I/O vector.

This equation can be rewritten as follows:

$$\sum_{i=1}^{m} P(A \cap E_i|F) = \sum_{i=1}^{m} \frac{P(A \cap E_i|F)P(F)P(E_i \cap F)}{P(F)P(E_i \cap F)} \tag{7}$$

$$= \sum_{i=1}^{m} \frac{P(A \cap E_i \cap F)P(E_i|F)}{P(E_i \cap F)} \tag{8}$$

$$= \sum_{i=1}^{m} P(A|E_i \cap F)P(E_i|F) \tag{9}$$

$$\approx \sum_{i=1}^{m} P(A|E_i)P(E_i|F) \tag{10}$$

The last equation holds since $P(A|E_i) \approx P(A|E_i \cap F)$: Usually there is only a very small number of I/O vectors which is valid in a given situation. If a sufficiently large I/O vector is taken, $E_i \approx F \cap E_i$ since most of the I/O vectors resulting from i edges are faulty most of the time.

The two remaining probabilities $P(A|E_i)$ and $P(E_i|F)$ can be estimated. The probability of

accepting an I/O vector with a given NDAAO under the condition that it is the result of i edges is denoted as $P(A|E_i)$. To get this probability, the following function is defined:

Definition 5.4: Number of transitions with i edges: The function

$$LTrans(x,i) = \sum_{\forall x' \in f(x)} \begin{cases} 1 & \text{if } |ES(\lambda(x),\lambda(x'))| = i \\ 0 & \text{else} \end{cases}$$

delivers the number of following states x' of a given state x which are reached by producing exactly i edges in the according evolution set.

A conservative estimate for $P(A|E_i)$ is the probability of accepting an I/O vector resulting from i edges from the NDAAO state with the *most leaving transitions with exactly i edges*:

$$P(A|E_i) \leq \frac{\max\limits_{\forall x \in X}(LTrans(x,i))}{\binom{m}{i}} \tag{11}$$

$\binom{m}{i}$ is the binomial coefficient of the number of controller I/Os m and the number of edges i. The binomial coefficient gives the number of possible I/O vectors resulting from i edges: In a given I/O vector there are $\binom{m}{i}$ possible ways to choose i I/Os and to alter them such that a new I/O vector is created. With $\max\limits_{\forall x \in X}(LTrans(x,i))$, the maximum number of transitions leaving one state in the NDAAO and producing exactly i edges is given. The coefficient gives the probability that an arbitrary I/O vector resulting from i edges is accepted in the NDAAO state with the most transitions with i edges.

It is also possible to replace $P(A|E_i)$ with the following equation instead of using the conservative estimate from Equation (11):

$$\overline{P}(A|E_i) \approx \frac{\sum\limits_{\forall x \in X} \frac{LTrans(x,i)}{|X|}}{\binom{m}{i}} \tag{12}$$

In this equation the *average* number of transitions leaving a NDAAO state and leading to i edges is calculated. This average number is then divided by the number of possible I/O vectors resulting from i edges. If each NDAAO state is equally likely to be the current state when a faulty I/O vector occurs, this represents the probability of accepting the I/O vector.

The second remaining probability in Equation (10) is $P(E_i|F)$. It denotes the probability that a faulty I/O vector is the result from i edges. It can be interpreted as the expected part of faults leading to an I/O vector with i edges. Since we only consider faults leading to a new I/O vector (deadlock faults are not considered in this work), it is obvious that the following equation holds:

$$\sum_{i=1}^{m} P(E_i|F) = 1 \tag{13}$$

Usually, the value for each $P(E_i|F)$ can only be roughly estimated. In the following, we assume that $P(E_i|F)$ can be conservatively estimated by an expert keeping to the following considerations: It is reasonable to expect a large portion of faults leading to only a few edges and to only have a small portion of faults leading to many edges (e.g. 80% leading to one edge, 10% leading to two edges etc.). This is based on the principle of Parsimony (Reiter 1987) where diagnosis is understood as a conjecture that some minimal set of components is faulty. Following

this principle, it can be assumed that a fault typically affects only a small number of components and thus I/Os.

The result of the preceding considerations are an upper bound and an expected average for the probability of accepting an I/O vector induced by a fault:

$$\sum_{i=1}^{m} P(A \cap E_i|F) \leq \sum_{i=1}^{m} \left(\frac{\max\limits_{\forall x \in X}(LTrans(x,i))}{\binom{m}{i}} \cdot P(E_i|F) \right) \tag{14}$$

$$\sum_{i=1}^{m} \overline{P}(A \cap E_i|F) \approx \sum_{i=1}^{m} \left(\frac{\sum\limits_{\forall x \in X} \frac{LTrans(x,i)}{|X|}}{\binom{m}{i}} \cdot P(E_i|F) \right) \tag{15}$$

The expert estimation of $P(E_i|F)$ tends to contain errors. The following considerations investigate the effects of $P(E_i|F)$ on the calculation of the measure from Equation (14). The aim is to give an upper bound for the measure that is independent from $P(E_i|F)$. The role of $P(E_i|F)$ in the calculation is to rate the different summands in Equation (14). Due to Equation (13), the sum of the values for $P(E_i|F)$ is one. Hence, it is obvious that Equation (14) results in its maximum value if the largest summand is rated with one and each other summand is rated with zero. The according worst case $P_{max}(E_i|F)$ is given by

$$P_{max}(E_i|F) = \begin{cases} 1 & \text{if } \forall j \neq i \ \frac{\max\limits_{\forall x \in X}(LTrans(x,i))}{\binom{m}{i}} > \frac{\max\limits_{\forall x \in X}(LTrans(x,j))}{\binom{m}{j}} \\ 0 & \text{else} \end{cases} \tag{16}$$

Probability $P_{max}(E_i|F)$ has the value one for the largest summand in Equation (14) and the value zero for each other summand. With this equation, it is possible to give the upper bound of the probability of accepting an I/O vector induced by a fault without using expert knowledge:

$$\sum_{i=1}^{m} P_{max}(A \cap E_i|F) \leq \sum_{i=1}^{m} \left(\frac{\max\limits_{\forall x \in X}(LTrans(x,i))}{\binom{m}{i}} \cdot P_{max}(E_i|F) \right) \tag{17}$$

For the example in Figure (10) the two measures $\sum_{i=1}^{m} P(A \cap E_i|F)$ and $\sum_{i=1}^{m} \overline{P}(A \cap E_i|F)$ are first calculated using expert knowledge for $P(E_i|F)$. The according upper bound independent from $P(E_i|F)$ will also be given. The values for $P(E_i|F)$ are defined a priori in Table 1. As a conservative estimate, we assume that only faults leading to one or two edges can occur. It is obvious that using the automaton in Figure 10, faults leading to three edges can certainly be detected since it does not have a transition with three edges. Since this portion of faults is also added to the expected portion of faults leading to one or two edges, Table 1 is a conservative estimate. In systems with many controller I/Os, it is especially conservative to assume a larger expected portion of faults with few edges: The weight of faults with many edges in the probabilities in Equation (14) and (15) is relatively low since the binomial coefficient $\binom{m}{i}$ of controller I/Os and edges becomes very large if $1 \ll i \ll m$.

It can be seen that state x_4 has the most leaving transitions with one edge. The result is

$$\frac{\max\limits_{\forall x \in X}(LTrans(x,1))}{\binom{3}{1}} P(E_1|F) = \frac{3}{3} \cdot 0.8 = 0.80 \tag{18}$$

For faults leading to two edges (here, x_1 must be considered) we get

$$\frac{\max\limits_{\forall x \in X}(LTrans(x,2))}{\binom{3}{2}} P(E_2|F) = \frac{2}{3} \cdot 0.2 = 0.13 \tag{19}$$

If Equation (20) and (19) are summed up, the result is $P(A \cap E_i|F) \leq 0.93$. This means that using the a priori knowledge from Table 1, there is an upper bound probability of about 93% for accepting an I/O vector induced by a fault. The result of the analysis independent from $P(E_i|F)$ show that the resulting probability is 100%. The largest summand from Equation (14) results from considering faults leading to one edge:

$$\frac{\max\limits_{\forall x \in X}(LTrans(x,1))}{\binom{3}{1}} P_{max}(E_i|F) = \frac{3}{3} \cdot 1 = 1 \tag{20}$$

If the automaton is in state x_4 and faults always lead to only one edge (resulting from the conservative determination of $P_{max}(E_i|F)$ in Equation (16)), the according fault cannot be detected since each possible behavior resulting from one edge can be reproduced.

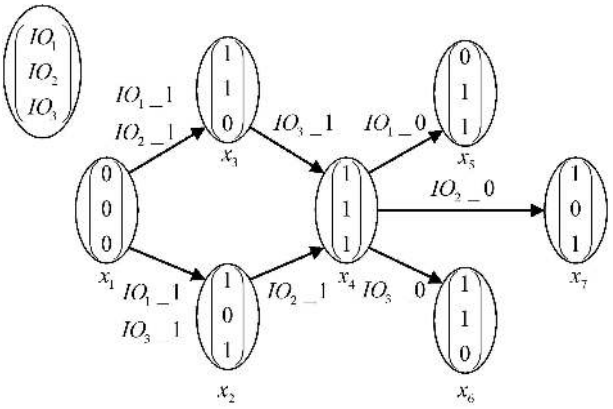

Figure 10. Example for the probability measures

number of edges i	1	2
$P(E_i\|F)$	0.8	0.2

Table 1. Definition of $P(E_i|F)$ for the example

The average probability can also be estimated (also assuming the values from Table 1 for $P(E_i|F)$). The result is

$$\overline{P} \approx \frac{\frac{5}{7}}{3} \cdot 0.8 + \frac{\frac{2}{7}}{3} \cdot 0.2 = 0.21$$

The estimate for the average probability of erroneously accepting a faulty I/O vector is about 21%. In the next section it will be shown that for existing systems the probability values are considerably lower than in the academic example.

6. Case Study

To demonstrate the relevance of the proposed method for existing closed-loop DES a case study is presented. The considered system is a laboratory facility at the institute of automatic control, University of Kaiserslautern, Germany. The purpose of the system depicted in Figure 11 is to treat work pieces that are stored in the feeder (left most station) with three machine tools. The system is controlled using a Siemens S300 PLC (Programmable Logic Controller) with 14 inputs and 15 outputs. The controller inputs and the corresponding sensors can be seen in Figure 11. Inputs that are written in *italic* (each input except inputs that are related to work piece position sensors) are connected to sensors with a specific technology that delivers a logical 0 if they detect something and a logical 1 if they do not detect anything. The outputs are given in Table 2. If they are set to 1, the according actuator gets activated. In order to make the I/O names easier to read they are not labeled IO_i but with I1.2 etc. to indicate the second ***i***nput that belongs to the first machine and O2.4 for the fourth ***o***utput of the second station.

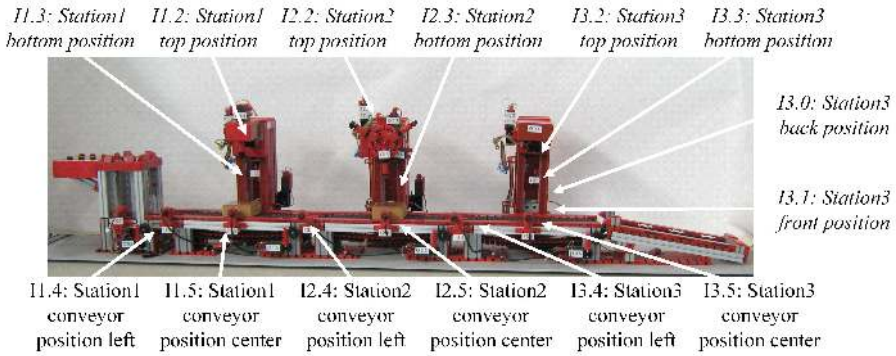

Figure 11. Case study system

A production cycle (or system evolution) consists of the treatment of two work pieces that are stored in the feeder. At the beginning of a system evolution the feeder pushes the first work piece to the conveyor. Then it is transported to the drilling machine. In the drilling station (station 1) the work piece is stopped and two holes are drilled. As soon as the first work piece has been treated in the first station it is transported to the vertical milling machine (station 2) and the next piece is taken from the feeder. The first work piece gets treated by the vertical milling machine (each of the three milling tools is applied to the piece) and the second one by the drilling station. After the treatment at the vertical milling machine is finished, the work piece gets transported to the horizontal milling station (station 3). The second work piece moves from the drilling station to the vertical milling machine. When the work piece has been treated in the horizontal milling machine, it is stored in the last station at the right side in Figure 11. This process continues until both work pieces have been treated by each of the three station.

For the application of the proposed diagnosis method it is necessary to have a model describing the fault-free behavior of the considered closed-loop system. For the case study system such a model can be identified using the method outlined in Section 4. The data base for the identification consists of 60 fault-free system evolutions (60 treatments of two work pieces). Before the identification algorithm can be applied, the identification parameter k has to be determined. As in the example in Figure 4 in Section 4.2, it is necessary to analyze I/O vector sequences

I/O	Description	I/O	Description
O1.2	drilling machine motor up	O2.2	vertical milling machine motor up
O1.3	drilling machine motor down	O2.3	vertical milling machine motor down
O1.4	drilling machine drilling motor	O2.4	vertical milling milling motor
O1.5	drilling machine conveyor on	O2.5	vertical milling machine conveyor on
		O2.7	change milling head
O3.0	horizontal milling machine motor back	O3.1	horizontal milling machine motor front
O3.2	horizontal milling machine motor up	O3.3	horizontal milling machine motor down
O3.4	horizontal milling machine milling motor	O3.5	horizontal milling machine conveyor on

Table 2. Controller outputs of the case study

of length two to distinguish non-equivalent closed-loop DES states with the same output. This can be determined a priori: The critical states are the situations when a work piece is between two position sensors like in the example in Section 4.2. It is thus possible to distinguish the states considering I/O vector sequences of length two. As a result of this consideration, k must be chosen to $k = 2$.

After the determination of k following the considerations of Section 4.2 it is necessary to analyze the collected system data and to check if the algorithm can be applied with $k = 2$. As explained in Section 4.3, the idea of the identification algorithm is to create an automaton state for each observed I/O vector sequence of length k. Since the identified model is used for online diagnosis purposes, it is important that it contains a state for each I/O vector sequence that can appear during the normal (fault-free) system behavior. If this is not the case, the model will always raise a false alert, when a new fault-free I/O vector sequence of length k appears which is not represented by an automaton state and can thus not be reproduced. Hence, it must be decided if each fault-free I/O vector sequence of length k has been seen when the identification data base has been collected. For this purpose it is useful to analyze the evolution of the cardinality of the observed languages of different length (referring to I/O vector sequences of different length). Figure 12 shows this evolution. It can be seen that the language of length two (L^2_{Obs}) converges to a stable level after about 14 observed system evolutions. From this it can be concluded that each fault-free I/O vector sequence of length two has been observed. The algorithm will thus create a state for each fault-free sequence.

In the identification algorithm, two states are connected if the underlying I/O vector sequences occur successively. A diagnosis model should represent each state transition which is possible during the fault-free system behavior to avoid false alerts. Hence, it is also necessary to state that the observed language of length $k + 1$ converges to a stable level. Figure 12 shows that $L^{k+1}_{Obs} = L^3_{Obs}$ converges to a stable level after about 40 system evolutions. It can thus be concluded that the identification algorithm creates each state transition which can occur during the fault-free system behavior.

Using the convergence of the observed language as a criterion for completeness of the system observation is a heuristic. It is clear that although the convergence is a strong indicator, it does not proof that there will not be new fault-free I/O vector sequences in future system evolutions. If there will be new sequences, they will be considered as fault-symptoms and lead to false alerts. It is thus necessary to choose k such that the according curve for $|L^{k+1}_{Obs}|$ shows a good

convergence. Figure 12 shows that for values larger than $k = 2$ ($n = k + 1 = 3$), the curves do not show a good convergence. From an engineers point of view, a small number of false alerts resulting from an non-perfect convergence is acceptable since the state estimation algorithm from Section 5.1 is capable of reinitializing the model for restarting diagnosis with an I/O vector sequence of length k at maximum. Our experience with the case study is that the reinitialization is usually much faster than k I/O vectors and most often only takes one vector.

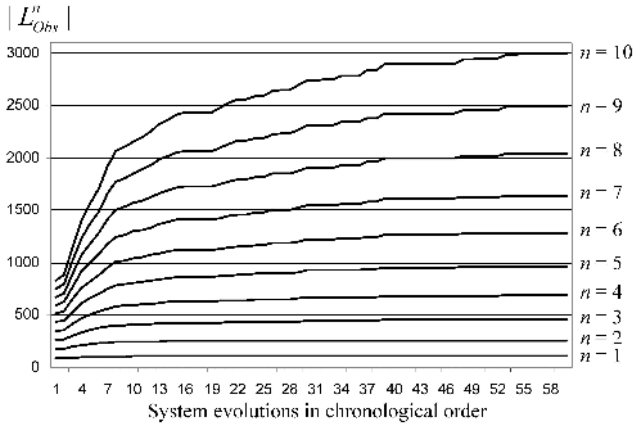

Figure 12. Evolution of the cardinality of the observed system language for the case study system

After the identification algorithm has been applied to the collected data with $k = 2$, a NDAAO with 121 states and 163 transitions is identified. A part of this automaton can be seen in Figure 13. In the figure, only the rising and falling edges between NDAAO states are depicted due to the size of the I/O vector.

Once a model has been identified, it can be evaluated with the probabilistic measures from Section 5.3. For the case study, the a priori probability $P(E_i|F)$ denoting the probability that a faulty I/O vector is the result from i edges is defined in Table 3. It is conservatively assumed that the portion of faults leading to an I/O vector with one edge is 80%. Following the consideration from Section 5.3, it is also conservative not to consider faults leading to more than three edges. Calculating the probabilities for accepting an I/O vector induced by a fault results in the following equations using the expert knowledge for $P(E_i|F)$.

$$\begin{aligned}\sum_{i=1}^{m} P(A \cap E_i|F) &\leq \frac{3}{\binom{29}{1}} \cdot P(E_1|F) + \frac{2}{\binom{29}{2}} \cdot P(E_2|F) + \frac{2}{\binom{29}{3}} \cdot P(E_3|F) \\ &\leq 0.084 \\ \sum_{i=1}^{m} \overline{P}(A \cap E_i|F) &\approx \frac{0.58}{\binom{29}{1}} \cdot P(E_1|F) + \frac{0.15}{\binom{29}{2}} \cdot P(E_2|F) + \frac{0.31}{\binom{29}{3}} \cdot P(E_3|F) \\ &\leq 0.016\end{aligned}$$

This shows that using the expert knowledge for $P(E_i|F)$ we get an upper bound of 8.4% and an average value of 1.6% of accepting an I/O vector induced by a fault.

If the upper bound is determined without expert knowledge for $P(E_i|F)$, the first step is to determine the largest summand in the calculation of $\sum_{i=1}^{m} P(A \cap E_i|F)$. The largest summand is $\frac{3}{\binom{29}{1}}$ for the state with the most leaving transition with one edge. As explained is Section 5.3, this summand is rated with one and all other summands are rated with zero in the next equation.

$$\sum_{i=1}^{m} P_{max}(A \cap E_i|F) \leq \frac{3}{\binom{29}{1}} \cdot 1 = 0.103 \tag{21}$$

The calculation shows that there is an upper bound independent from the expert knowledge for $P(E_i|F)$ of 10.3% for accepting an I/O vector induced by a fault.

The low probabilities show that the identified model is appropriate for online diagnosis. They guarantee that there is *at most* a probability of 10.3% (or even 8.4% using some expert knowledge) for missing a fault. The calculated average value of 1.6% indicates that the real probality can be assumed to be much smaller than the upper bound.

number of edges i	1	2	3
$P(E_i\|F)$	0.8	0.15	0.05

Table 3. Definition of $P(E_i|F)$ for the case study

To show how faults can be detected and isolated based on the identified model, a fault at the sensor that is connected with the controller input $I3.3$ is treated as an example. The fault that has been introduced artificially is a short circuit that causes the sensor to switch its signal from 1 to 0. The fault has been introduced when the first work piece was treated by the second machine (vertical milling). At the same time, the second work piece gets drilled in the first station. The short circuit has been produced when milling with the second of the three tools in station 2 started and the milling head left the top position. This situation is depicted in Figure 13. After the observation of $I2.2\_1$ indicating that the milling head just left its home position, the evaluator algorithm delivers state x_{18} as single state estimate. Hence, no fault is detected. Then the falling edge $I3.3\_0$ is observed that occurs due to the fault. The evaluator algorithm does not find a following state of x_{18} that can be reached by this edge. Since $\widetilde{X}_t$ is empty, a fault is detected. Consequently the residuals are applied with $\widetilde{x} = x_{18}$ as the former unique estimate.

$$Res1(\widetilde{x}, u(t)) = \{I3.3\_0\}$$
$$Res3(\widetilde{x}, u(t)) = \{\}$$

$Res1$ delivers the unexpected edge that is caused by the fault. $Res3$ results in an empty set. It can be seen that $Res1$ isolates the faulty I/O $I3.3$.

Generally, applying the residuals does not always lead to sets only containing one element. In Roth (2010) more examples can be found which show that although the residuals often deliver several possibly faulty I/Os, the resulting sets are usually small in comparison with the complete I/O vector.

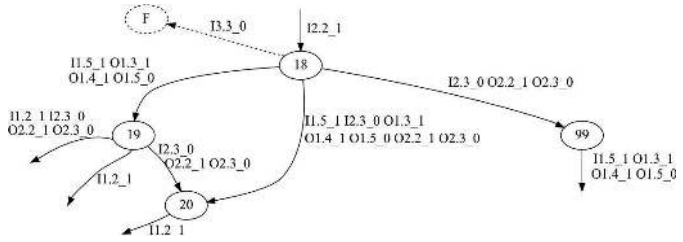

Figure 13. Example for a fault in the case study system

7. Conclusions and Outlook

In this paper a diagnosis method based on an identified model has been presented. It has been explained how the necessary data for identification and for online diagnosis can be captured in a typical industrial setup. The parameterization of a known identification algorithm has been shown to build a useful model for online diagnosis. The resulting models are fault-free models which represent the nominal system behavior. To assess the fault detection capability of an identified model, two probabilistic measures have been introduced. They estimate the probability of accepting a faulty behavior as fault-free. To isolate faults, a method based on residuals for DES has been presented.

Current work investigates timed system behavior in the model identification and fault isolation procedures.

References

Cassandras, C. G., Lafortune, S., 2006. Introduction to Discrete Event Systems. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Fanti, M. P., Seatzu, C., 2008. Fault diagnosis and identification of discrete event systems using Petri nets. Proceedings of the 9th International Workshop on Discrete Event Systems, WODES 08. Göteborg, Sweden, pp. 432-435.

Danancher, M., Roth, M., Lesage, J.-J., Litz, L., 2011. A comparative study of three model-based FDI approaches for Discrete Event Systems. 3rd International Workshop on Dependable Control of Discrete Systems (DCDS'11), Saarbuecken, Germany, pp. 29-34.

Dotoli, M., Fanti, M. P. , Mangini, A. M., Ukovich, W., 2009. On-line fault detection in discrete event systems by Petri nets and integer linear programming. Automatica 45 (11), pp. 2665-2672.

Hashtrudi Zad, S., Kwong, R. H., Wonham, W. M., 2005. Fault diagnosis in discrete-event systems: Incorporating timing information. IEEE Transactions on Automatic Control 50 (7), pp. 1010-1015.

Isermann, R., Balle, P., 1997. Trends in the application of model based fault detection and diagnosis of technical processes. Control engineering practice 5 (5), pp. 709-719.

Izadi, I., Shah, S. L., Shook, D. S., Chen, T., 2009. An Introduction to Alarm Analysis and Design. Proceedings of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, Spain, pp. 645-650.

Klein, S., Litz, L., Lesage, J.-J., 2005. Fault detection of discrete event systems using an identification approach. Proceedings of the 16th IFAC World Congress, Paper Mo-E17-TO4, Prag.

Klein, S., 2005. Identification of Discrete Event Systems for Fault Detection Purposes. (PhD thesis), Shaker Verlag.

Pandalai, D., Holloway, L., 2000. Template languages for fault monitoring of timed discrete event processes. IEEE transactions on automatic control 45 (5), pp. 868-882.

Moor, T., Raisch, J., Young, S., 1998. Supervisory control of hybrid systems via l-complete approximations. Proceedings of the IEE fourth Workshop on Discrete Event Systems WODES 98, Cagliari, Italy, pp. 426-431.

Reiter, R., April 1987. A theory of diagnosis from first principles. Artificial Intelligence 32 (1), pp. 57-95.

Roth, M., Lesage, J.-J., Litz, L., 2010. Identification of Discrete Event Systems - Implementation Issues and Model Completeness. Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics (ICINCO) Funchal, Portugal.

Roth, M., 2010. Identification and Fault Diagnosis in Industrial Closed-Loop Systems. (PhD

thesis), Logos Verlag, ISBN 978-3-8325-2709-9.

Roth, M., Lesage, J.-J., Litz, L., 2009. A residual inspired approach for fault localization in DES. Proccedings of the 2nd IFAC Workshop on Dependable Control of Discrete Event Systems (DCDS'09), Bari, Italy, pp. 347-352.

Russell, E. L., Chiang, L. H., Braatz, R. D., 2000. Fault detection in industrial processes using canonical variate analysis and dynamic principal component analysis. Chemometrics and Intelligent Laboratory Systems 51, pp. 81-93.

Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., Teneketzis, D., 1996. Failure diagnosis using discrete-event models. IEEE transactions on control systems technology 4 (2), pp. 105-124.

Sayed-Mouchaweh, M., Philippot, A., Carre-Menetrier, V., 2008. Decentralized diagnosis based on Boolean discrete event models: application on manufacturing systems. International journal of production research 46 (19), pp. 5469-5490.

Schneider, S., Litz, L., Danancher, M., 2011. Timed residuals for fault detection and isolation in discrete event systems. 3rd International Workshop on Dependable Control of Discrete Systems (DCDS'11), Saarbuecken, Germany, pp. 35-40.

Estrada-Vargas, A.-P., Lopez-Mellado, E., Lesage, J.-J., 2010, A comparative analysis of recent identification approaches for discrete-event systems, Mathematical Problems in Engineering, Vol. 2010, Article ID 453254, doi:10.1155/2010/453254.

Meda-Campana, M.-E., 2002, On-line identification of discrete event systems: fundamentals and algorithms for the synthesis of Petri net model, (PhD thesis), Centro de Investigacion y de Estudios Avanzados del Instituto Polit´ecnico Nacional, Unidad Guadalajara, Mexico.