Distributed identification of concurrent discrete event systems for fault detection purposes

Matthias Roth, Jean-Jacques Lesage, Lothar Litz

Distributed identification of concurrent discrete event systems for fault detection purposes

Matthias Roth, Jean-Jacques Lesage *Member, IEEE,* Lothar Litz *Member, IEEE*

***Abstract*-An identification method for concurrent discrete event systems (DES) is presented. The purpose of the identified model is model-based fault detection in closed loop industrial DES such as production or manufacturing systems. The method consists of identifying an automata network with a scalable restriction of the network behavior. We especially deal with incomplete observation of the system language within a finite time horizon which is a typical result of a high degree of concurrency. It is shown how models identified on the basis of an incompletely observed language can be used for fault detection.**

I. Introduction

Fault Detection and Isolation (FDI) in Discrete Event Systems (DES) has been an active research area in the last 10 to 15 years. Since the introduction of the diagnoser approach in [1] various modifications and improvements of this model-based diagnosis technique have been developed. The core of this method is to build component models of a system that reflect the normal, fault-free behavior as well as the behavior in the case of given faults. The goal is to detect and diagnose a fault in the system by only considering so called "observable events" using a special observer (diagnoser). Although the method has successfully been applied to small and medium sized systems, the manual model-building is a laborious task if large industrial production systems are considered. Using identification techniques to get the necessary models for fault detection and isolation (FDI) allows minimizing the modeling effort. In contrast to the classical diagnoser approach an identified model is usually based on the fault-free observed system behavior. Thus, the identified model contains only the fault-free behavior. If the identified model is used as an observer it must be analyzed if an observed behavior cannot be reproduced by the fault-free model. In this case a fault is detected.

Since industrial production systems often consist of concurrent parts an identification method should be able to handle this concurrency. The identification of concurrent discrete event systems has been examined by several research groups in the last years. Usually, the methods are based on Petri Nets. In [2] an identification algorithm for Interpreted Petri Nets is given that detects concurrent transitions by the analysis of the order of their occurrence. The algorithm uses some rules when to add non-measurable places to the identified net such that the observed sequence of transitions and measurable net places can be produced by the identified model.

In [3] an identification method for Petri nets based on the formulation of algebraic constraints for an integer programming problem is given. The method allows considering additional information such as structural constraints if available. Since like shown in [3], solving the integer programming problem has exponential complexity with respect to the longest string in the observed language, finding a solution for large scale systems with long cycles may be impossible within a reasonable time.

Dealing with the identification of concurrent systems one key problem arises that has not been addressed in literature so far: the concurrency can lead to a very large possible behavior of the system that cannot be completely observed within a given time. In order to deal with incomplete observation of the system behavior a new distributed identification approach is presented in this paper. It is an extension of an identification method given in [4] that yields a monolithic automaton that is especially suitable for fault detection.

II. Monolithic Identification of a DES

The identification approach in [4] considers closed loop DES. The closed loop consists of controller and plant, which is a typical configuration for industrial production systems. The identification works on the basis of the I/O (input/output) vectors of the controller collected during p different production cycles.

Definition 1: The j-th I/O vector in the h-th of p production cycles is defined as $u_h(j) = (I_1(j), ..., I_s(j), O_1(j), ..., O_m(j))_h$ with I_1, .., I_s and O_1, .., O_m denoting the considered inputs and outputs of the controller of the closed loop system.

Definition 2: If during the h-th production cycle, l_h I/O vectors have been observed, the sequence is denoted as $\sigma_h = (u_h(1), u_h(2), ..., u_h(l_h))$.

If each observed I/O vector is considered as a letter of an alphabet it is possible to define the set of observed words with length q observed during p different production cycles. A word represents an observed I/O vector sequence.

Matthias Roth is with the Institute of Automatic Control at the University of Kaiserslautern, Germany, and the LURPA, Ecole Normale Supérieure de Cachan, France. (mroth@eit.uni-kl.de)

J.-J. Lesage is the head of the LURPA at the Ecole Normale Supérieure de Cachan, F - 94235 Cachan Cedex, France (lesage@lurpa.ens-cachan.fr)

L. Litz is the head of the Institute of Automatic Control at the University of Kaiserslautern, D – 67653 Kaiserslautern, Germany (litz@eit.uni-kl.de)

Definition 3: The observed words of length *q* are denoted as

$$W_{\mathrm{Obs}}^{q} = \bigcup_{i=1}^{p}\left(\bigcup_{j=1}^{l_i - q+1} \left(u_i(j), u_i(j+1), ..., u_i(j+q-1)\right)\right)$$

With this definition it is possible to describe the behavior of a system by the language built on the basis of the observed words.

Definition 4: The observed language of length *n* is

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

The identification algorithm given in [4] consists of building an automaton that generates the observed language of length *n* of the identified DES. Since a system with interaction of controller (deterministic behavior) and physical process (non-deterministic behavior) has to be identified, the identified model is a non-deterministic autonomous automaton with output (NDAAO):

Definition 5: NDAAO = $(X, \mathbf{\Omega}, r, \lambda, \mathrm{x}_0)$

X finite set of states,

$\mathbf{\Omega}$ output alphabet,

$r : \mathbf{X} \rightarrow 2^X$ non-deterministic transition relation,

$\lambda : \mathbf{X} \rightarrow \mathbf{\Omega}$ output function,

$x_0 \in X$ initial state.

The output alphabet $\mathbf{\Omega}$ consists of the observed I/O vectors such that in the case of an identified NDAAO $\mathbf{\Omega} = W_{Obs}^{1}$. λ assigns a word and thus an I/O vector from the output alphabet to each state.

The NDAAO generates a set of words of length *n* from each state $x(i) \in X$ that is defined as:

Definition 6:

$W_{x(i)}^{1} = w \in \mathbf{\Omega}^1 : w = (\lambda(x(i)))$ and

$W_{x(i)}^{n>1} = \{w \in \mathbf{\Omega}^n : w = (\lambda(x(i)), \lambda(x(i+1)), ...,$

$\lambda(x(i+n-1))) : x(j+1) \in r(x(j)) \forall\ i \le j \le i+n-2\}$

The definition of the language of length *n* generated by an NDAAO follows definition 4:

Definition 7: The language of length *n* generated by an NDAAO is $L_{Ident}^{n} = \bigcup_{i=1}^{n} \bigcup_{x \in X} W_{x}^{i}$

Thus, the language consists of all the words up to length *n* that can be produced starting from each state $x \in X$.

The algorithm in [4] allows to construct an NDAAO on the basis of observed words of the parametric length *k*, the identified automaton is able to produce the observed language of the system and is *k*+1-complete ($L_{Obs}^{k+1} = L_{Ident}^{k+1}$) [5].

If the language (and thus the cardinality of the word set) of the system has not been completely observed like in the case of the dashed line in Fig 1, the identified NDAAO will not be able to produce each fault-free I/O vector sequence of length $n=k+1$ of the system. Not yet observed regular sequences are not part of the model and can thus not be produced which leads to false alerts when monitoring the system with the identified model.

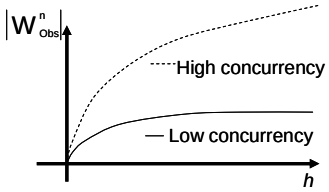

Fig 1: Evolution of the number of observed words of length *n* over production cycles *h*

For large scale systems with a high degree of concurrency it is by experience not possible to observe all fault-free words (I/O vectors) within a reasonable time. It is only possible to determine an upper bound of new words that are expected in each new system cycle (see as an example Fig 7). To handle concurrent systems which make a complete observation impossible, a new distributed approach is presented.

III. Proposed Distributed Approach

A. Automata Network Behavior

The main goal of the presented approach is to handle incomplete observation due to a high degree of concurrency of a closed loop DES when identifying a model. To handle the concurrent parts of a system it is advantageous to divide the system into concurrent subsystems for the identification and online observation. Even if the combined behavior of the concurrent subsystems can not be completely observed within a reasonable time, a complete (or almost complete) observation is much more likely for each single subsystem. For the considered class of closed loop industrial production systems this division can be made by considering well-defined parts of the I/O vectors of the controller for each subsystem.

Definition 8: The *j*-th *partial* I/O vector in the *h*-th of *p*-th production cycle is defined as $u_h(j) = (I_1(j), ..., I_s(j), O_1(j), ..., O_m(j))_h$ with I_1, .., I_s and O_1, .., $O_m \in \{0,1,-\}$. The I/Os get a "-" if they are not considered in the partial vector, else they have values "1" or "0" according to the observed values.

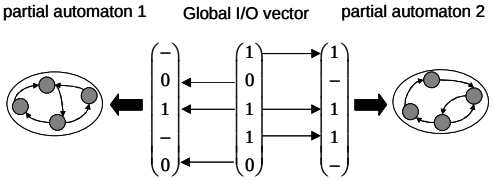

Fig. 2: Division of an I/O vector into two subsystems

In this paper we suppose that the division of the system into subsystems and thus the allocation of I/Os to the subsystems is performed using a-priori knowledge. Other approaches to automatically divide the system based on an analysis of the observed language are part of current research.

For each subsystem a partial automaton is identified

according to the procedure given in [4]. Each partial automaton contains only parts of the entire I/O vector like in Fig 2.

The combined output of two or more partial NDAAOs is defined by a join operation on the outputs of the active states:

Definition 9: The join of two partial I/O vectors (v_1,v_2) is defined as

$$J(v_1,v_2)(i)=\begin{cases} c \text{ if } |v_1|\neq|v_2| \\ c \text{ if } v_1(i)\neq -\wedge v_2(i)\neq -\wedge v_1(i)\neq v_2(i) \\ v_1(i) \text{ if } v_2(i)=- \\ v_2(i) \text{ if } v_1(i)=- \\ v_1(i) \text{ if } v_1(i)=v_2(i) \end{cases}$$

$\forall i=1,..,\max(|v_1|,|v_2|)$

The I/Os at each position of the two partial I/O vectors are compared. If one of them is "-" (it is not considered in the subsystem represented by the partial automaton) the value of the other vector is taken. If one of them is "1" and the other "0" *c* for contradiction is taken.

From this definition follows directly that a valid state combination $(x_1,..,x_n)$ of partial automata for modeling of a system state must fulfill the condition $c\notin J(\lambda(x_1),..,\lambda(x_n))$. If the identified automata are used to monitor the system they can evolve independently as long as their state combination produces a valid system output. The language of the automata network is the language of the cross product of the partial automata given as:

Definition 10: Cross product of two partial NDAAO:

State space:

$X_{\|}=X_1\times X_2 \mid \forall(x_1,x_2)\in X_1\times X_2 : c\notin J(\lambda(x_1),\lambda(x_2))$

State output

$\forall(x_1,x_2)\in X_{\|} : \lambda((x_1,x_2))=J(\lambda(x_1),\lambda(x_2))$

Transition relation

$\forall(x_1,x_2)\in X_{\|} : r((x_1,x_2))=\{(x_{p1},x_{p2}) \mid$

$(x_{p1},x_{p2})\in(x_1\cup r(x_1))\times(x_2\cup r(x_2)) : c\notin J(\lambda(x_{p1}),\lambda(x_{p2}))$

and $(x_{p1},x_{p2})\neq(x_1,x_2)\}$

Initial state: $x_0=(x_{1\_0},x_{2\_0})$

From this definition it follows directly that the cross product contains only states with a valid output ($\forall x\in X_{\|} : c\notin\lambda(x)$).

Let B_S denote the complete fault-free system behavior and $B_{S,i}$ the fault-free partial behavior of the *i*-th subsystem. It is obvious that $B_{S,1}\|..\|B_{S,n}\supseteq B_S$. Since there is usually a part $(B_{S,1}\|..\|B_{S,n})\setminus B_S\neq\{\}$ the cross product is not suitable for use in FDI. Some faulty behaviors $B_{S,F}\in(B_{S,1}\|..\|B_{S,n})\setminus B_S$ cannot be distinguished from fault-free behaviors using the cross product even if for each identified partial behavior $B_{S,i,Ident}=B_{S,i}$ holds.

Instead of a proof consider the following (extreme) example: Each I/O in the global I/O vector can be assigned to an own partial automaton containing only this I/O. The resulting network of partial automata can produce every combination of I/O values since the automata are not synchronized. Hence, no matter which I/O vector sequence will be the consequence of a fault, it is part of the identified network behavior and thus the fault can not be detected. Even if in real systems such an extreme choice of subsystems would not be made, it shows that the network behavior must be restrained in order to use the network for fault detection.

To deal with this problem the framework in Fig 3 is proposed. In order to reduce the set $(B_{S,1}\|..\|B_{S,n})\setminus B_S$ containing non-detectable faults a *permissive* observed cross product is introduced that is built on the basis of the observed cross product of the partial automata. This automaton is used to detect if the combination of the identified partial behaviors $(B_{S,1,Ident}\|..\|B_{S,n,Ident})$ is part of the already observed fault-free behavior $B_{S,Obs}$ of the entire system. Using a tolerance specification it is possible to adjust the amount of combined but not yet observed partial behavior $(B_{S,1,Ident}\|..\|B_{S,n,Ident})\notin B_{S,Obs}$ that is accepted as fault-free. The tolerance specification can be constructed using the gradient at the end of the curve in Fig 1. The gradient shows how many new words are usually expected in a new cycle. This information can be translated into an automaton defining the threshold of consecutive new words that are considered as a fault symptom.

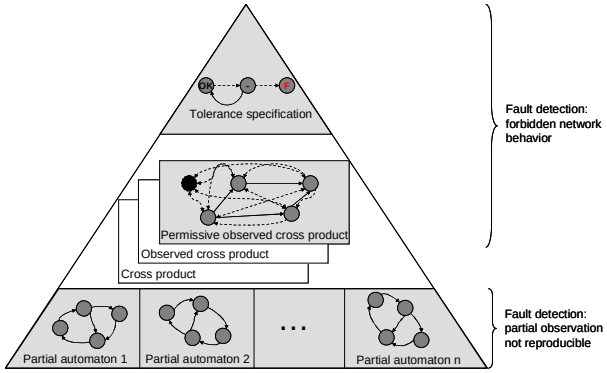

Fig. 3: Proposed model schema

B. *Restriction of the Automata Network Behavior*

The first step to restrict the automata network behavior is the introduction of the observed network behavior:

Definition 11: The Observed Cross Product of *n* partial NDAAO is a 5-tuple $(X_{obs\|},\Omega_{obs\|},r_{obs\|},\lambda_{obs\|},x_{obs\|\_0})$ according to definition 5. The output alphabet consists of the possible state combinations of the network of *n* partial automata ($\Omega_{obs\|}=X_1\times..\times X_n$). The output function assigns a state combination of the network to each state of the observed cross product.

The observed cross product is derived from the cross product of the partial automata and data of the observed production cycles:

Algorithm 1: $ObsX := \{\}$

For each recorded production cycle $\sigma_h = (u_h(1), u_h(2), ..., u_h(l_h))$:

$\forall (x_{\|}(1), x_{\|}(2), .., x_{\|}(l_h)) \mid W^{l_h}_{x_{\|}(1)} = \sigma(h)$:

$ObsX := ObsX \cup (x_{\|}(1), x_{\|}(2), .., x_{\|}(l_h))$

end_for

For each state $x_{\|,nObs} \in X_{\|} \mid x_{\|,nObs} \notin ObsX$:

$\forall x_{\|} \in X_{\|} : r(x_{\|}) := r(x_{\|}) \setminus x_{\|,nObs}$

$X_{\|} := X_{\|} \setminus x_{\|,nObs}$

end_for

For each $x_{\|Obs} \in X_{\|}$: $\lambda_{obs\|}(x_{\|obs}) := (x_1, x_2, ..x_n) \in$

$X_1 \times X_2 \times .. \times X_n$ from step "state space" in definition 10

end_for

$X_{obs\|} := X_{\|}, r_{obs\|} := r_{\|}$

End algorithm 1.

The algorithm consists in "playing" the cross product using the observed I/O sequences (Fig 4). Each state that is not necessary to produce the observed I/O sequences is removed from the state set of the cross product. The transition relations are adapted such that they contain only remaining states. The remaining cross product is transformed into an observed cross product by changing the output function of the states. The output function $\lambda_{obs\|} : \mathbf{X}_{obs\|} \rightarrow \mathbf{\Omega}_{obs\|}$ assigns to each state the combination of the partial automata states that has been used in the step "State space" in definition 10 to build the cross product state.

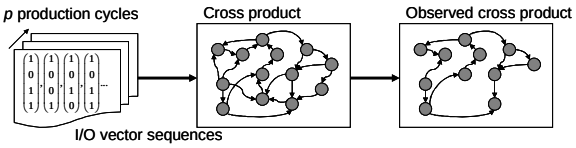

Fig 4: Construction of the observed cross product

The automata network and the observed cross product can now be combined according to the following definition:

Definition 12: Combined evolution of *n* partial automata and observed cross product: Let $x_{obs\|\_actual}$ be the actual state of the observed cross product with $\lambda_{obs\|}(x_{obs\|\_actual}) = (x_{1\_i}, .., x_{n\_i})$. The *n* partial automata can move from state combination $(x_{1\_i}, .., x_{n\_i})$ to another state combination $(x_{1\_j}, .., x_{n\_j}) \neq (x_{1\_i}, .., x_{n\_i})$

if $\forall\ 1 \le a \le n$: $x_{a\_i} = x_{a\_j}$ or $x_{a\_j} \in r_a(x_{a\_i})$ holds (where *a* is the index of the *a*-th of *n* partial automata) and

if $\exists x_1 \in X_{obs\|} \mid \lambda_{obs\|}(x_1) = (x_{1\_j}, .., x_{n\_j}) \wedge x_1 \in r_{obs\|}(x_{obs\|\_actual})$. If the condition is fulfilled the observed cross product moves to the new actual state x_1 and the automata network changes to the state combination $(x_{1\_j}, .., x_{n\_j})$.

With this definition, the network can only produce two consecutive state combinations $(x_{1\_i}, .., x_{n\_i})$, $(x_{1\_j}, .., x_{n\_j})$ if they are part of the observed cross product. In order to deal with incomplete observation like the dashed line in Fig 1, it is necessary to allow a certain number of not yet observed state combinations to reduce the number of false alerts. To do this the permissive observed cross product and the tolerance specification are introduced in the next two subsections.

C. Permissive Observed Cross Product

The aim of the permissive observed cross product (POCP) is to make it possible that the automata network can produce not yet observed trajectories of state combinations (and thus I/O vector sequences), but to recognize such a situation. Hence, a new function is introduced that determines if a transition of the POCP has already been observed during the model learning phase.

Definition 13: The permissive observed cross product (POCP) is a 6-tuple $(X_{POCP}, \Omega_{POCP}, r_{POCP}, \lambda_{POCP}, x_{POCP\_0}, \Theta_{POCP})$. The first five elements are defined according to definition 5. $\Theta_{POCP} : X_{POCP} \times X_{POCP} \rightarrow \{true, false\}$ is the transition observation function.

The POCP is constructed starting from the observed cross product according to the following algorithm:

Algorithm 2:

Step 1: Enlarged output alphabet

$\Omega_{POCP} := \Omega_{obs\|} \cup e$ where *e* is the empty letter.

Step 2: Basic state space

$X_{POCP} := X_{obs\|}$

Step 3: Basic transition relation

$r_{POCP} := r_{obs\|}$

Step 4: Transition observation function:

$\forall x \in X_{POCP} : \Theta_{POCP}(x, r(x)) := true$

Step 5: Enlarged state space :

$X_{POCP} := X_{POCP} \cup x_e \mid x_e : \lambda_{POCP}(x_e) := e, r_{POCP}(x_e) := \{\}$

$r_{POCP}(x_e) := X_{POCP} \cup x_e$ where x_e is a state with an empty output that is connected to each state in the state space.

Step 6: Enlarged transition relation

$\forall (x \neq x_e) \in X_{POCP} : r_{POCP}(x) := X_{POCP} \setminus x$

Step 7: Complete definition of the observation function:

$\forall x \in X_{EOCP}$: if $\Theta_{POCP}(x, r_{POCP}(x))$ not yet defined:

$$\Theta_{POCP}(x, r_{POCP}(x)) := false$$

End algorithm 2

Fig 3 shows the result of the algorithm: the POCP includes the complete observed cross product plus one state x_e (the black circle) containing the empty symbol as output. Each state is connected to the state x_e and to each other state by a transition. If this transition has been observed – it was already part of the observed cross product – the transition observation function has the value "true" which is represented by a solid line in Fig 3. If the transition was

added during the construction of the POCP the transition observation function has the value “false” (dashed line in Fig 3).

The dynamics of the combination of automata network and POCP is defined as follows:

Definition 14: Let x_{POCP_actual} be the actual state of the POCP. The *n* partial automata can move from one state combination $(x_{1\_i},..,x_{n\_i})$ to another state combination $(x_{1\_j},..,x_{n\_j}) \neq (x_{1\_i},..,x_{n\_i})$ if $\forall\, 1 \le a \le n:\ \ x_{a\_i} = x_{a\_j}$ or $x_{a\_j} \in r_a(x_{a\_i})$ holds. The POCP evolutes according to the following rule:

If $\exists x_{POCP} \in X_{POCP} \mid (\lambda(x_{POCP}) = (x_{1\_j},..,x_{n\_j})) \wedge\ x_{POCP} \in r_{POCP}(x_{POCP\_actual})$ the POCP moves to the according state

else the POCP moves to the state with the empty output x_e ($x_e \in r_{POCP}(x_{POCP\_actual})$ is always true because of step 5 and step 6 of the algorithm 2).

Following this definition, the network can produce observed and unobserved trajectories of state combinations. If two consecutive state combinations have already been observed, the POCP will take a transition with $\Theta_{POCP} = true$. If the consecutive state combinations are not yet known, a transition with $\Theta_{POCP} = false$ will be taken. The state with the empty output aggregates all state combinations that have not been observed. Hence, if the automata network produces state combinations that are unknown they can be represented by this state. By analyzing the transition observation function Θ_{POCP} it is now possible to distinguish observed and unobserved state trajectories of the network.

D. Tolerance specification

The information of Θ_{POCP} during the combined evolution of network and POCP can now be used to specify the length of yet unobserved network trajectories that is considered as a fault symptom. This specification is given by a tolerance specification automaton.

Definition 15: The tolerance specification is a 6-tuple $(X_{Tolerance}, \Omega_{Tolerance}, r_{Tolerance}, \lambda_{Tolerance}, x_{Tolerance\_0}, \Theta_{Tolerance})$ according to definition 13. The output alphabet $\Omega_{Tolerance}$ consists of the three elements $\{OK, Fault, -\}$ where *OK* represents a fault free situation, *Fault* represents a situation that leads to a fault detection and “-“ represents an undecided situation.

The tolerance specification is to be used in combination with the network and the POCP. The evolution of network and POCP is given in definition 14. The tolerance specification automaton monitors the POCP evolution via the transition observation function:

If the POCP moves from a state x_A to a state x_B the tolerance specification moves from its actual state x_1 to a state x_2 iff $x_2 \in r_{Tolerance}(x_1)$ and $\Theta_{Tolerance}(x_1, x_2) = \Theta_{PCPA}(x_A, x_B)$. Reading the output of the following state x_2 it is possible to see if the network evolution that led to the PCOP trajectory is considered as a fault ($\lambda_{Tolerance}(x_2) = Fault$) or not.

An example for a design of the tolerance specification is given in Fig 7 in the next chapter. The combined evolution of tolerance specification and POCP works on the basis of observed (solid line) and unobserved (dashed line) transitions. If the POCP takes an already observed transition (solid line in Fig 3) the tolerance specification also takes an observed transition (solid line in Fig 7). If the transition in the POCP has not yet been observed (dashed line in Fig 3) the tolerance specification takes an unobserved transition (dashed line in Fig 7). In order to produce the observed and the unobserved behavior defined in the tolerance specification, the automata network, POCP and the tolerance specification (grey parts in Fig 3) evolve according to the rules given in this chapter.

IV. Application

In order to evaluate the ability of the identified models to detect faults the method was not directly applied to an industrial production system. For an evaluation of the method it is necessary to do tests with faults in the system which is usually not possible in a production system in operation. The lab size plant which was treated has a significant degree of concurrency that can also be found in industrial manufacturing systems. The evolution of the observed language is also comparable with industrial systems that have been treated with the method from [4].

The plant in Fig 5 has 30 binary controller I/Os. During one production cycle the plant in Fig 5 treats three work pieces. Since there are three machine tools in the plant that work concurrently, a complete observation of the whole system language takes very long. In Fig 6 it can be seen that the evolution of $|W^n_{Obs}|$ representing the set of I/O vector sequences up to length three does not reach a stable level and is thus not completely observed. A monolithic automaton identified according to section II with the parameter *k*=2 has 416 states. *k*=2 is chosen such that each physical system state can unambiguously be distinguished from each other physical system state by the analysis of *k*=2 consecutive I/O vectors. Hence, each automaton state represents only one physical system state which is advantageous for FDI. Since the set of words with length 3 is not completely observed the basic set of the identification is supposed to grow within the next production cycles. Consequently, monitoring the system with the monolithic model led to false alerts as soon as new words of length 3 have been observed. Fault-free words that are not observed within the given 62 fault-free production cycles are not included in the model and lead to fault detection since the model is not able to produce them.

To deal with this problem the system has been divided

into three overlapping subsystems (I/O groups) that can be seen in Fig 5 using apriori knowledge. Note that an intersection of the groups is not necessary. Even for high values of n $|W^n_{Obs}|$ can be considered as completely observed for each subsystem as the graphs converge very early to a stable level (they are not depicted due to space limitations).

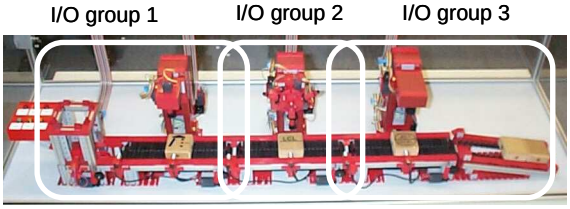

Fig 5: Identified lab system

With the available data of 62 production cycles a partial NDAAO for each I/O group has been identified with k=2. The partial automata have 15, 15 and 17 states. Then, the observed cross product has been built. It consists of 327 states. Following the algorithm given in section III.C the POCP has been constructed.

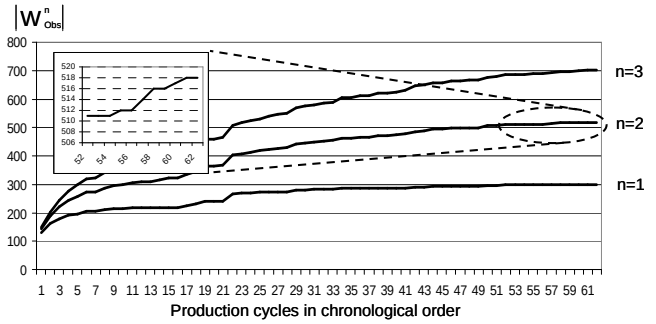

Fig 6: Evolution of the number of observed words of length n

In Fig 6 it can be seen that in each new production cycle after the 53th one at most two new words of length n=2 are observed in a new cycle (e.g. in the last one). This information can be interpreted as follows: in a fault-free production cycle we expect at most two consecutive unobserved transitions in the POCP since the partial automata are expected to produce at most two yet unobserved consecutive state combinations. The yet unobserved state combinations produce the new sequences of I/O vectors. Hence, an appropriate tolerance specification should accept two unobserved transitions as fault free but should lead to fault detection after the occurrence of the third unobserved transition in the POCP. Fig 7 shows the tolerance specification designed for the lab plant. As soon as the first unobserved transition is detected, the *OK*-state is left. If only one or two yet unobserved transitions are observed it is possible that the tolerance specification moves back to the "*OK*"-state. Here, the design is heuristically chosen such that each way back from a "-"-state to the *OK*-state takes one already observed transition more than it took unobserved transitions to get to the "-"-state. After the occurrence of three consecutive unobserved transitions in the POCP, the tolerance specification will be in the *Fault*-state.

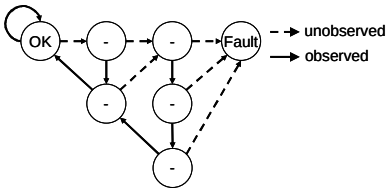

Fig 7: Tolerance specification for the lab plant

Using the automata network, the POCP and the tolerance specification allowed monitoring the whole system with an identified model without the occurrence of false alerts whereas built-in faults have been detected. Note that faults can be detected due to two situations: Firstly, a partial automaton can not produce the observed partial behavior. Secondly, the network behavior exceeds the accepted behavior and the tolerance specification moves to the *Fault*-state. The combination of POCP and tolerance specification realizes a scalable restriction of the automata network behavior.

The proposed framework is also able to handle inaccuracy during the I/O allocation to the subsystems which may not always be as obvious as in the presented case. Even if sensors of e.g. I/O-group 3 have been assigned to I/O-group 1, we were able to detect the built-in sensor-faults. Faults in cases like this are often not detectable by the partial automata but by the tolerance specification.

V. Outlook

Current research is focused on building I/O groups of the concurrent subsystems based on the observed process behavior. The aim is to provide a method that divides a process like in Fig 5 automatically in concurrent parts and to allocate the according I/Os. Another research area is the development of fault localization algorithms for the proposed framework.

References

[1] M. Sampath, R. Sengutpa, S. Lafortune, K. Sinnamohideen, D.C. Teneketzis, "Failure Diagnosis using Discrete-Event Models", *IEEE Trans. on Control Systems Technology*, Vol. 4, No. 2, pp. 105-124, March 1996.

[2] M.E. Meda-Campaña; E. Lopez-Mellado: "Identification of Concurrent Discrete Event Systems using Petri Nets". "*2005 IMACS: Mathematical Computer, Modelling and Simulation Conference*". July 2005, Paris

[3] A. Giua, C. Seatzu, "Identification of Petri nets via integer programming", *CDC-ECC'05: 44th Int. Conf. on Decision and Control and European Control Conference* (Seville, Spain), pp. 7639 – 7644, December 2005.

[4] S. Klein, J.-J. Lesage, L. Litz, "Fault detection of Discrete Event Systems using an identification approach", *16th IFAC World Congress*, CDROM paper n°02643, 6 pages, Praha(CZ), July 4-8, 2005.

[5] T. Moor, J. Raisch, and S. O'Young, "Supervisory control of hybrid systems via l-complete approximations", *in Proc. of the IEE fourth Workshop on Discrete Event Systems WODES'98*, Cagliari, Italy, August 1998, pp. 426-431.