ISSN: 2277-9655
Impact Factor: 1.852

IJESRT

INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

A New Tool of Surveillance for Discrete Event Systems

Mohamed Fri*1, Fouad Belmajdoub2

*1,2University of Sidi Mohamed Ben Abdellah, Faculty of Science and Technology, Laboratoire Techniques Industrielles, Road Imouzzer B.P. 2202 Fez, Morocco

Mohamed.fri@usmba.ac.ma

Abstract

In this paper, we discuss the different types of faults for discrete event systems that lead to a system malfunction, afterwards we illustrate the principle of control of discrete event systems, and at the end we present an algorithm that controls the start and the end of each action intending to ensure that each step runs are in their permitted time range, this algorithm is easy and effective to perform the proper functioning of discrete event systems.

Keywords: Discrete Event System; Supervision; Surveillance; Defects

Introduction

Discrete event systems (DES) are systems with finite state space where transitions are driven by discrete events (communication networks, database systems, traffic networks, digital circuits and manufacturing systems). A control theory of a general class of discrete event systems was initiated by Ramadge and Wonham [14]. Control-theoretic concepts such as controllability and observability have been formalized in the DES setting.

The diagnostic of industrial processes is a scientific discipline that aims at the detection of faults in industrial plants, their isolation, and finally their identification. Its main task is the diagnosis of process anomalies and faults in process components, sensors and actuators. Early diagnosis of faults that might occur in the supervised process renders it possible to perform important preventing actions. Moreover, it allows one to avoid heavy economic losses involved in stopped production, the replacement of elements and parts [16].

Discrete event systems (DES) formalisms are largely applied in the industrial automation area, in order to develop powerful methods to design controllers and diagnostic algorithms. The scientific community focused on proposing efficient DES methods to design supervisory controllers, fault tolerant controllers, as well as fault detection and isolation algorithms [2, 5, 7, 10, 15]. In this context, fault detection and diagnosis of DES received considerable attention in the past years, motivated by the practical need of ensuring the correct and safe functioning of complex industrial systems. In particular, in the industrial automation field, the problem of representing systems in their ‘‘complexity containment’’ under nominal and faulty situations is crucial [16].

In this paper we present an algorithm which controls the start and the end of each action intending to ensure that each step runs in its time range permit, this algorithm is effective and easy to perform the proper functioning of discrete event systems.

The rest of this paper is organized as follows: Section 2 provides a review of discrete event systems. Section 3 presents terminologies and references, and Section 4 presents signal and system faults. Section 5 provides a review of permanent and intermittent faults. Section 6 presents control-monitoring module. Section 7 formulates our algorithm, and finally, section 8 draws conclusions.

Discrete Event Systems

The discretization of calculations in a computer has served as an inspiration to many researchers. It was not long before people realized that many systems (especially digital systems) could be successfully modelled as Discrete Event Systems (DES). Such systems are those where events (changes of state) happen spontaneously, are logically ordered relative to each other, and are not tied to a continuous global time.

An example of a DES is the high-level model of a vending machine. The machine has a number of discrete states, defined by how many articles are available inside the machine and how many coins are inserted. A change of state happens

ISSN: 2277-9655
Impact Factor: 1.852

when a coin is inserted or when the machine delivers an article. These changes of states are named "events". Some events can happen only in a given state. For example, the machine will not deliver an item if there are no goods loaded, or if the correct amount of money is not inserted. Naturally, DES models can be applied to much more complex systems, such as manufacturing cells [13].

Discrete-Event Systems can be formally modelled using many different approaches, ranging from Petri nets to Markov chains, fuzzy matrixes [10], and modal logic [12, 14]. However, the most commonly used method is the representation through automata, and for all practical purposes-Finite-State Machines (FSM). Besides being a very intuitive approach, this also allows for the application of results from Automata Theory, which is a well-studied area.

An FSM is a five-tuple $G = (\Sigma, Q, \delta, q_0, Q\ f)$, where Σ is a finite set of symbols (and is often called the alphabet), Q is a finite set of states, δ is a partial transition function $\Sigma \times Q \rightarrow Q$, q_0, is the initial state of the system, and $Q\ f \subseteq Q$, is a subset of the states, which are defined to be "final" (a final state is sometimes also referred to as a "marked state"). The special "empty" symbol ε, which does not belong to Σ, is used to denote the empty string (i.e., the string of length zero). The notation Σ^* stands for the set of all strings of symbols from Σ and ε. The transition function δ can be naturally extended to the partial function δ': $\Sigma^* \times Q \rightarrow Q$, where $\delta'(\sigma, q) = \delta(\sigma, q)$, for all $\sigma \in \Sigma$ and $q \in Q$ and $\delta'(\sigma s, q) = \delta'(s, \delta(\sigma, q))$, for $s \in \Sigma^*$, $\sigma \in \Sigma$, and $q \in Q$.
Usually, δ' is denoted by δ and is used instead of the original transition function. An FSM can be interpreted as a DES if states are considered to be states of the system and transitions labeled with symbols from Σ are considered to be events happening in the system. Strings of symbols would describe sequences of events.

The language L(G) is defined to be the set of all possible sequences of events in the system. The FSM G is said to generate L(G). The language Lm(G) is defined to be the set of all sequences of events which lead to a final state. The FSM G is said to accept Lm(G). The generated language L(G) is always a superset of Lm(G). More formally,

$L(G) = \{s \mid s \in \Sigma^*, \delta(s, q_0) \text{ is defined}\}$,
$Lm(G) = \{s \mid s \in \Sigma^*, \delta(s, q_0) \text{ is defined}, \delta(s, q_0) \in Q\ f\}$,
and $Lm(G) \subseteq L(G)$.

The prefix-closure of a language is defined to be the set of all prefixes of strings in the language. The empty string ε is a prefix of any string. For all automata, prefix closing the generated language produces a language equal to the generated language itself. More formally,

$\bar{L} = \{s \mid s \in \Sigma*, \exists t \in \Sigma*, st \in L\}, \overline{L(G)} = L(G)$.

A prefix-closed language is a language which equals its prefix-closure. Prefix closure is an important operation because it describes all the possible partial behaviors of a DES. An example of a DES is the simplified model of a customer at a store Fig. 1.

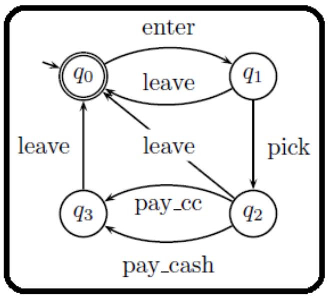

$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[rr, bend left, "enter"] & & q_1 \arrow[ll, bend left, "leave"] \arrow[dd, "pick"] \\
\\
& q_3 \arrow[uu, "leave"] & & q_2 \arrow[uull, "leave"'] \arrow[ll, bend right, "pay\_cc"'] \arrow[ll, bend left, "pay\_cash"]
\end{tikzcd}$$

Figure 1: DES model of customer in a store

The customer can enter the store, pick something to buy, pay with cash or a credit card, and leave at any time. Here Σ = {"enter", "pick", "pay-cash", "pay-cc", "leave"}. The set of states is $Q = \{q_0, q_1, q_2, q_3\}$. The transition function can be determined from the diagram in Fig.1, e.g., $\delta(pick, q_1) = q_2$. The initial state is marked with q_0. This state is the only final state, as well (i.e., $Q\ f = \{q_0\}$). Examples of event sequences are "enter, leave" or "enter, pick, pay cc". The second sequence is not "complete"-it does not belong to Lm. However, it belongs to $\overline{Lm}$, since it is a prefix of the sequence "enter, pick, pay cc, leave", which is in Lm.

The examples presented here are very simple, in such a way that anyone can easily imagine the application of DESs in factory processes, computer protocols, and other areas. This is why scientists are increasingly becoming interested in the DES paradigm.

Terminologies and References

This section presents some usual definitions of important terminologies in the Supervision and Control domain. There is no consensus about the terminologies definition presented in this section. Hence, the objective of the paper not to propose new terminologies, instead, it presents definitions used to describe ours researches. Different authors contribute to this section, but a concentrate of definitions can be found in [3] and [13].Control, monitoring and supervision are first defined and then the definition of other terms used in this paper is given.

Control: triggers the execution of a set of operations by giving orders to the process actuators, which may be:

- A set of operations corresponding to the manufacturing sequence of the product.
- A set of operations executed in order to restore the process functionality offered during normal execution.
- Actions with a high priority level applied in order to protect the shop workers and to prevent catastrophic developments.
- Some checking, tuning or cleaning operations executed in order to maintain the process in an operational state.

This means that our definition of control includes all the functions actually acting on the process.

Monitoring: collects data from the process and from the controller, determines the actual state of the controlled system and makes the inferences needed to produce additional data (historic, diagnosis, etc.). Monitoring is limited to data processing and has no direct action on the models or on the process.

Supervision: computes and sets the parameters of the control sequence to be executed according to the state of the control system and to the state of the process. This includes normal and abnormal operations. During normal operation, supervision takes the decisions to raise the indecision in the control system (real-time scheduling, optimization, control sets and switching from one control law to another). When a process failure occurs, supervision takes all the decisions necessary to allow the system to resume normal operation (rescheduling, recovery actions, emergency procedures, etc.). It should be noted that supervision takes place in a hierarchical structure (of at least two levels). At the lowest level only the control and monitoring functions are generally implemented, no real decisions have to be taken.

Fault: Action, voluntary or not, that does not take all the specifications into account.

Defect: Difference between the actual value of a parameter and its nominal value.

Error: Part of a model which does not exactly match the specifications of the physical system. Logically, an error is the consequence of a fault.

Latent error: The error is qualified as latent as long as the erroneous part of the model has not been used. After using the erroneous part of the model, the error becomes effective.

Failure: Event characterizing a situation in which an operation is not executed by a resource because its state no longer corresponds to the nominal specifications.

Breakdown state: State of a resource from which the system cannot provide the specified service. This state is the consequence of a failure.

Symptom: Event or data by which the detection system identifies an abnormal process operation. The symptom is the only information the monitoring system knows at the detection step.

Recovery point: State reachable from the breakdown state in which the system must be driven to resume normal operation.

Recovery sequence: Set of ordered actions executed to bring the process back from the breakdown state to the recovery point.

According to these basic concepts, we can define the elementary functions of the supervision and monitoring system. Between brackets the letter M, S or C indicates to which previously discussed group (Monitoring, Supervision, and Control) the function belongs.

Detection (M): determines the normality or abnormality of the functioning system. Two classes of abnormal operations are considered:

- The first includes situations in which basic operating constraints of the process are violated (collisions for instance).
- The second one groups together situations in which the part routing (control law) is not respected (fabrication delays for instance).

Follow (M): maintains the state space of the system, it traces the events observed in the control/supervision model to update the state of system.

Diagnosis (M): looks for a causality link between the observed symptom, the failure and its origin. Classically, three sub-functions are distinguished:

- Localization determines the subsystem responsible for the failure,
- Identification identifies the causes of the failure,
- Explanation justifies the conclusions.

Prognosis (M): foresees the consequences of a failure on the future operation of the system. The consequences can be immediate (resource unavailable) or induced (faulty parts in the workshop).

Decision (S): determines the state that must be reached to resume to normal operation, then determines the sequence of corrective actions to be performed to reach this state.

Recovery (C, S): acts both on the process by changing the states of the resource or equipment and on the control system by changing the control laws, the part routing, etc. Three classes can be defined:

- Minor, only the control laws are adapted,
- Significant, other resources are reallocated,

ISSN: 2277-9655
Impact Factor: 1.852

- Major, reallocated resources need to be prepared to execute the recovery.

Signal and System Faults

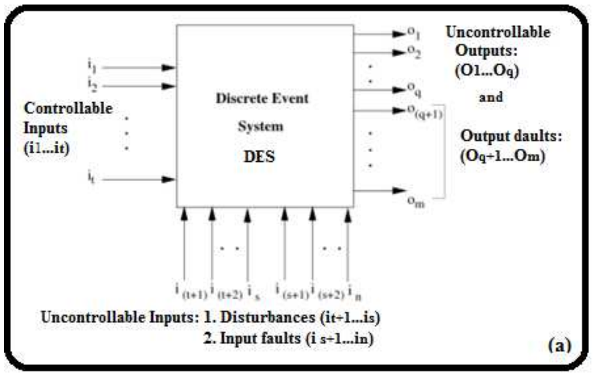
The kinds of faults which can occur in the type of discrete event systems we consider include the stuck-signal faults, and the system or equipment faults.

Stuck-signal faults

These faults occur when any of the actuators or sensors in the system, owing to mechanical, electrical, or electromagnetic interference problems, gets stuck in a particular position with its logic status becoming either true or false permanently, until a recovery occurs through a repair or a replacement. When an actuator gets stuck in the on/off position such fault signals are denoted by so(stuck open)/sc(stuck closed) respectively; where the associated fault events are denoted by soF/scF and the recovery events by soR/scR, respectively. In the tank system the filling tap t1 may be prone to a stuck open fault signal denoted by t1so, with the fault event denoted as t1soF and the recovery event denoted as t1soR. When the fault event t1 soF occurs, filling will continue to occur even after the command to switch off the tap has been given, unless the fault recovery event occurs.

When a sensor gets stuck in the up/dn(down) position such fault signals are denoted by sup(stuck up)/sdn(stuck dn) respectively; whereas the associated fault events are denoted by supF/sdnF, and the recovery events by supR/sdnR, respectively. The tank system in Figure .3 has a level sensor n which may be prone to a stuck up fault signal, denoted by nsup.

This signal has two events: the fault event nsupF, and the corresponding recovery event nsupR.
It should be noted that stuck-signal faults are a type of output signals since they are dependent on values of signals prone to stuck-signal faults.
Referring to Figure .2, it can be seen that of the m output signals, m - q are fault signals.



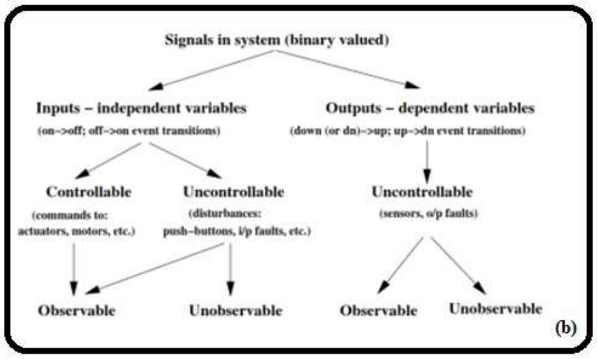

Signals in system (binary valued)
Inputs – independent variables
(on->off; off->on event transitions)
Outputs – dependent variables
(down (or dn)->up; up->dn event transitions)
Controllable
(commands to: actuators, motors, etc.)
Uncontrollable
(disturbances: push-buttons, i/p faults, etc.)
Uncontrollable
(sensors, o/p faults)
Observable
Unobservable
Observable
Unobservable
(b)

Fig.2. Input-Output view of a discrete event system

System/equipment faults

Apart from faults of the signals there are faults of systems and its components. Certain fault signals such as equipment failures, power disruptions, system software crashes, etc., affect the entire system. These can occur spontaneously in the system depending only on their flown values, not those of any other signal in the system. They are thus independent variables and form part of the inputs to the system.

In the tank system of Figure .3, a leakage fault signal, which causes the fluid levels to drop in the tank, is an example of a system/equipment fault. The events of the leakage fault are leakageF and leakageR, denoting leakage fault and recovery from leakage events.

It should be noted that system faults are a type of input signals since they are independent of values of other signals. Referring to Figure .2, it can be seen that of the n input signals, n - s are fault signals.

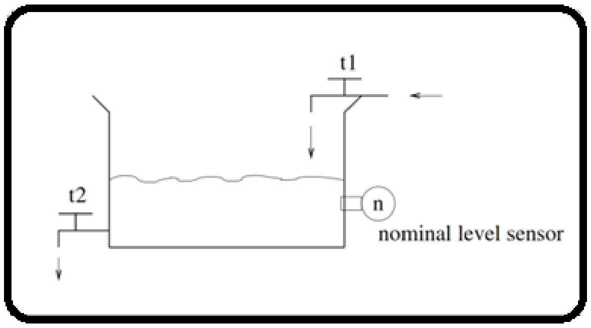


Fig.3. Tank system schematic

Permanent and Intermittent Faults

Another categorization of faults arises from the manner in which faults are reset after they occur.

Permanent faults

If the recovery event occurs only due to a re-pair/replacement of the fault, then the fault is regarded as a permanent fault.

Intermittent faults

If the recovery event can occur either spontaneously or through repair/replacement, then the fault is regarded as an intermittent fault. Example is a loose wire that makes and breaks contact spontaneously.

It is important to distinguish between these two types of faults, since the intermittent fault spontaneous recovery events, which tend to be uncontrollable and unobservable, may the system to oscillate between non-faulty and fault states. Permanent faults, on the other hand, are associated with recovery events (repair/replacement) which are controllable and observable, and the system cannot spontaneously move from a fault state to a non-fault one.

Control-Monitoring Module

Rapid technological advances in computer science have offered a wide range of possibilities to design control architectures. There are many basic architecture propositions, but centralized, hierarchical and "heterarchical" (as well as bionic, holonic, fractal, etc.) are the most commonly accepted. In this paper, we will use the term heterarchical to indicate an architecture that does not have the same characteristics as centralized or hierarchical architecture. The heterarchical architecture is adopted in order to pursue full local autonomy in which the global information is minimized or eliminated. This implies that:

1. External higher levels of control can change according to the activity to be coordinated. In this case one module can compose, for instance, two different hierarchical structures.
2. The communication between entities will not, necessarily, have a master/slave relationship, for example, they can co-operate, negotiate or dynamically change roles from master to slave and vice-versa.
3. We can introduce a new entity or modify the existing ones without significant structural changes.

These flexibilities introduced by the heterarchical architecture imply a more complex relationship between modules, risking failure situations hardly detected during the design process. We observe too that these flexibilities increase the difficulty of solving failure situations if their origins are not local, as the modules often do not know to where the faulty treatment functions must be propagated. This second problem does not exist either in hierarchical nor in centralized architecture, thus, no other reference, as far as the authors know, takes into account that new situation. Part of our contribution is to prevent that situation including local information about the modules' relationship.

This research is based on the modular organization presented in [15], called Control-Monitoring architecture. In their approach only resource failures are taken into account. The control and monitoring system is considered error free. When a resource failure occurs, the corrective actions to be executed are performed according to the activity state (function in execution, resource used, kind of manufactured products and production strategy specified by the user). Failure processing is not limited to the classical sequence (detection, diagnosis, decision and recovery).

The Acquisition/Routing block manages all these functions, as shown in figure .4. This block is based on an algorithm, which directs incoming messages to the most suitable functions, according to the nature of the data and the state of the monitoring system. Moreover, this algorithm maintains the state of this model and triggers the suitable monitoring, control and/or supervision functions [15].

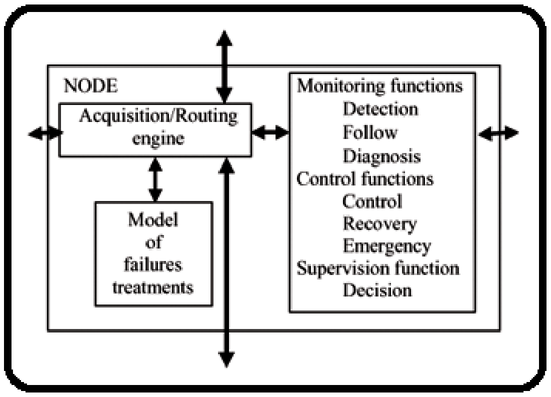


Figure .4: A generic module for control, supervision and monitoring

Two complementary tools are used to specify this approach. Petri nets with Objects are used for modeling control, recovery and emergency sequences and failure treatment. An extended entity relationship model provides a process representation (called Information System) in which data, which is not easy to model by means of Petri nets (time notions, histories, data flow, etc.) can be found.

In a sense, our research considers the topics pointed out previously to build a systematic procedure for distributing a centralized model of supervision and control. It is based on the Petri nets (PN), as described in [17], and shows the advantage that each part of the control process has, at least, the same properties as the whole model and at most all good properties (boundedness, liveness, home state).

This procedure is founded on linear PN invariants theory and the results of applying it is a set of sub-models, each one describing the behavior of one resource (or a set of them) and its interactions with other resources. Some sub-models have redundant information about the systems, because the relation between resources must be represented in all entities that use its services. The method is detailed in the following sections.

Algorithm

Each system has a role played in the form of a cycle that repeats steadily and containing several steps, the total time of the cycle is called the execution time and each step is intended in a well-defined time interval. If any of these steps exceeds the stated period thing that may be caused by the malfunction of one of the sensors or actuators or a work accident...etc. Then it will be a duty to intervene to correct the problem. For this reason, we thought to develop an algorithm to check whether each step respects the execution time. This concept is formalized and generalized in the following algorithm:

```
1.  Public class surveillance {
2.  Public static void main (arg[] stag)
3.  {Long θ, t, α;
4.  Date θ[] = θ[N-1];
5.  Boolean I[] =I[N-1];
6.  Date tmax[] = tmax[N-1];
7.  Boolean O[] =O[N-1];
8.  For (int I; i<N; i++)
9.  { while (t≤ θ[i])
10. { if (Ii==1) continue T1; }
11. pro=1;
12. T1: α= θ[i] – t;
13. for (int j=I; j<N;j++)
14. { tmax [j] = tmax [i]-α; }
15. While (t<=tmax)
16. {if (O[i] ==1) continue T2;}
17. prob=1;
18. T2: α= tmax[i] – t;
19. for (j=i+1; j<N; j++)
20. { θ[j] = θ[j]-α;  }}}}
```

To implement our algorithm, we must measure or compute the time interval for each step, this time interval contains the time of normal operation and the safety time. The latter is required in case of change of the unauthorized load or the environment and the amortization of the system ... etc. The safety time is not always used (or required), in other words there are steps that do not use this time or benefit only from one part.

After performing the measurements for each step, we define the start time earlier tmin(i) the start time later θ(i) and the end time later tmax(i) .

In our algorithm waiting for the action (i) should start before the start time later θi. Information of beginning action (i) is given by the input variable I(i):

If I(i) = 1 before time θ(i) , we deduce a value α(α= θ(i) - t) which represents the difference between the start time later step(i) and real time of the step(i) beginning.

After subtracting the value of α from all start time later θ(j) values of all steps having not yet run.
If the clock t value exceeds the θ(j) and Ii is not yet activated our algorithm enforce output "prob" to "1". This will shut down the system or trigger an alarm according to the choice of the user.

After verifying of the step (i) had a good start in the expected range we verify thereafter the end time later for this step.

If O(i) = 1 before time tmax, we deduce a value α (α = tmax(i) - t) which represents the difference between the start time later step(i) and real time of the step (i) start.

After subtracting the value of α from all end time later tmax(j) values of all steps having not yet run.

If the clock t value exceeds the tmax(i) and O(i) is not yet activated our algorithm enforce output "prob" to "1". This will shut down the system or trigger an alarm according to the choice of the user.

Characteristics of surveillance methods based model

There are several methods of surveillance based model in the literature. These methods are based on a model of normal behavior and/or failed system.

The actual observation of the current state of the system, about surveillance, is compared with the estimated by the model to detect a fault condition. Our surveillance methods ensure the following:
. The surveillance system is easy to implement
. The surveillance system detect defects at the earliest possible time
. The surveillance system is achievable in real time
. The surveillance system is conceivable algorithmically

Conclusion

The technological evolution of real systems and in particular control systems has greatly facilitated the monitoring tasks performed to ensure maximum efficiency of operation.
In this paper, we have brought a constructive contribution to this technology by focusing our study

on the preparation and presentation of an algorithm that allow the use and operation of sequential systems.

This algorithm discussed and studied in this phase has a positive impact on the efficiency and effectiveness of discrete event systems and especially the systems of control when monitoring spots sequential action.

References

[1] *Ashvin Radiya, G. Robert Sargent, "A logic-based foundation of discrete event modeling and simulation", ACM Transactions on Modeling and Computer Simulation '4(1), pp: 3–51, January 1994.*

[2] *C.G. Cassandras, S. Lafortune, "Introduction to discrete event systems (2nd ed.)", New York, NY, USA: Springer, 2008.*

[3] *M. Combacau, P. Berruet Charbonnaud, A. Khatab, E. Zamai, "Supervision and monitoring of production systems", in: Proc. MCPL'2000, Grenoble, july 2000*

[4] *M. Combacau, E. Zama, A. Chaillet-Subias, "Monitoring Strategies as Control Structure of Monitoring Architectures Based on Discrete Event Systems", Computational Engineering in System Applications, Nabeul-Hammamet, Tunisia, April1998*

[5] *M. Dotoli, M.P. Fanti, A.M. Mangini, W. Ukovich, "On-line fault detection of discrete event systems by Petri nets and integer linear programming", Automatica '45, pp: 2665–2672, 2009.*

[6] *Feng Lin, Hao Ying, "Modeling and control of fuzzy discrete event systems", IEEE Transactions on Systems, Man, and Cybernetics, Part B '32(4), pp: 408–415, August 2002.*

[7] *C.N. Hadjicostis, "Finite-state machine embed dings for non-concurrent error detection and identification", IEEE Transactions on Automatic Control '50(2), pp: 142–153, 2005.*

[8] *K. Patan, "Artificial Neural Networks for the Modelling and Fault Diagnosis of Technical Processes", Springer Pub, 2008.*

[9] *J. C. Laprie "Dependability basic concepts and terminologies", Springer Verlag edition, ISBN: 82296-8, 1992.*

[10] *J. Lunze, "Fault diagnosis of discretely controlled continuous systems by means of discrete-event models", Discrete Event Dynamic Systems: Theory and Applications '18(2), pp: 181–210, 2008.*

[11] *P. J. Ramadge, W. M. Wonham, "The control of discrete event systems", Inproceedings of the IEEE, Vol. 77, pp: 81-98, January 1989.*

[12] *P.J. Ramadge and W.M. Wonham, "Supervisory control of a class of discrete-event systems", SIAM Journal on Control and Optimization '25, pp: 206–230, 1987.*

[13] *S. C. Lauzon, A. K. L. Ma, J. K. Mills, B. Benhabib, "Application of discrete-event system theory to flexible manufacturing", IEEE Control Systems Magazine '16(1), pp:41–48, February 1996.*

[14] *S. L. Ricker and K. Rudie, "Know means no: Incorporating knowledge into discrete-event control systems", IEEE Transactions on Automatic Control '45(9), pp:1656–1668, September 2000.*

[15] *M.Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, D. Teneketzis, "Diagnosability of discrete-event systems", IEEE Transactions Automatic Control '40, pp: 1555–1575, 1995.*

[16] *A. Tilli, A. Paoli "Rule-based compos able modelling of industrial automation automata under nominal and faulty conditions. In: Proceedings of seventh IFAC symposium on fault detection, supervision and safety of technical processes, Barcelona, Spain, 30June 3 July, 2009*

[17] *R. Valette, B. Pradin-Ch zalviel, F. Girault "An introduction to Petri net theory". Fuzziness in Petri nets Eds , 1999.*