A methodology for modular model-building in discrete automation

Matteo Sartini[⋆,1]*, Andrea Paoli[⋆,1], Richard C. Hill[†], Stéphane Lafortune[◦,2]
[⋆]CASY - Center for Research on Complex Automated Systems, University of Bologna
Viale Carlo Pepoli 3/2, IT40136 Bologna, ITALY
{matteo.sartini, andrea.paoli}@unibo.it
[†]Deptartment of Mechanical & Manufacturing Engineering, University of Detroit Mercy
4001 W. McNichols Road, Detroit, MI 48221-3038, USA
hillrc@udmercy.edu
[◦]Deptartment of Electrical Engineering and Computer Science, University of Michigan
1301 Beal Avenue, Ann Arbor, MI 48109-2122, USA
stephane@eecs.umich.edu

Abstract

Our objective is to develop a general and versatile approach for building structured formal models of complex automated systems in order to facilitate their control and diagnosis. For this purpose, we present a methodology that builds the complete model of a system by composing models of the individual hardware components, their physical coupling, and the associated control logic. We choose to employ a hierarchical decomposition that separates the control logic into a high level that manages the sequence of control actions and a low level that implements the control actions. The low level is composed of control logic and physical components (sensors and actuators) grouped into a device. In order to capture the physical constraints between the components in a device, we propose the notion of a physical constraint automaton, *which is composed with the generic component automata to generate the complete model of the device. We also show how the methodology allows the introduction of component faults into the overall model. The effectiveness of the proposed approach is demonstrated on a micro flexible manufacturing system.*

1 INTRODUCTION

Advances in industrial automation are leading to systems that are increasingly complex. Designing control logic that provably satisfies given specifications for these systems is a challenging task. The role of this control logic includes correctly managing and coordinating all system devices to achieve the desired functionality, monitoring the system for fault detection and isolation, and ensuring the overall quality of the work produced. Recent work in industrial automation has focused on the concepts of modularity and reusability of the control logic [1, 3, 17]. To achieve all these objectives, the concepts and techniques of discrete event system theory are beginning to receive attention in industrial automation for control logic synthesis. Specifically, techniques from supervisory control, fault diagnosis, and fault-tolerant control of discrete event systems [8, 10] can be employed to verify whether or not a given specification is realizable or the faults of a given system can be diagnosed. Existing theory can also assist in the synthesis of the monitoring and control logic [6, 9, 12].

All of the above techniques, however, presume the availability of a complete formal model of the system; building such a model is a difficult task due to its dependence on deep knowledge of the system components and their physical coupling. To reduce the complexity of modeling the overall system in a monolithic manner, researchers have considered decentralized approaches and decomposition methods [5, 15]. A fault detection and isolation approach is additionally presented in [5]. Specifically, this paper presents an algorithm to construct a diagnostic automaton to detect non-nominal behaviors during system operation. This automaton is constructed based on knowledge of the system, and fault isolation is implemented by a diagnostic procedure where the system is excited by system-dependent forcing commands. In [15], a boolean discrete event model-based approach for fault detection and isolation of manufacturing systems is presented. The work claims that the well-known problem of state-space explosion can be avoided by considering simple system modules and approaching the diagnosis problem using a fault-free model-based approach: each behavior that does not correspond to the modeled one is considered a faulty behavior.

*Corresponding author

[1]The research of the first and second authors is in part supported by MIUR and in part supported by the European Artemis Joint Undertaking funded project CESAR.

[2]The research of the fourth author is supported in part by US National Science Foundation grant ECCS-0624821.

In this paper, we address the challenges of model building by presenting a general and versatile methodology for building in a modular manner the complete model of a complex automated system starting from individual components and their physical coupling. In addition, we define post-fault behavior in order to improve the precision of fault detection and isolation. We employ finite-state automata as our modeling formalism and propose a hierarchical decomposition that separates the control logic into a high level which manages the sequence of control actions and a low level which implements the control actions. This hierarchical decomposition enables reuse of the generic models of the low-level components. In [7] a related architecture in the context of supervisory control problems is presented. The low-level models incorporate low-level control actions as well as fault detection logic for component faults. This fault detection logic requires modeling the faulty behavior of the components in addition to their fault-free behavior.

The focus of this paper is on model-building at the lower level of the proposed hierarchy. We start from generic fault-free models of low-level components such as actuators and sensors. In order to capture physical constraints among these low-level components in a given automated system, we propose the notion of a *physical constraint automaton*, which is then coupled with the generic component automata by parallel composition. This approach achieves the desired characteristics of modularity, composability, and reusability. We also demonstrate how faulty behavior can be gradually incorporated into the model by incrementally enhancing the generic component models to include faults and by adjusting the associated physical constraint automata in a manner that captures the effects of these faults on the physical coupling. As a consequence, the entire faulty behavior is obtained by parallel composition of the individual faulty models. The steps of our modeling methodology are presented using a micro flexible manufacturing system. We then use this example to illustrate how formal discrete event system theory can be applied to the models to prove that the given control specification can be achieved and the faulty behavior of the system can be diagnosed.

This paper is organized as follows. We present in Section II a general overview of the proposed architecture for the control logic. We then describe in Section III the associated modular model-building methodology for fault-free and faulty behavior. Section IV describes how to use the obtained models for formal verification of system properties using discrete event system theory. Section V is a brief conclusion.

2 AN ARCHITECTURE FOR SUPERVISORY CONTROL IN INDUSTRIAL AUTOMATION

In complex automated systems such as manufacturing systems, it is imperative to design the supervisory controller and hence the control logic to achieve modularity and reusability. A crucial point in this regard is the separation of actuation mechanisms from control policies in order to hierarchically manage the plant. In other words, control should be viewed as the composition of: (i) a set of basic actions; and (ii) a set of coordination policies for the execution of these actions. This is the approach adopted and further elaborated on in [3] and [4]. Designing the control logic using a hierarchical strategy supports component interoperability and reusability and facilitates diagnosis and reconfiguration. The proposed ideas are applied to part of a real system that will be serve as a testbed throughout this work.

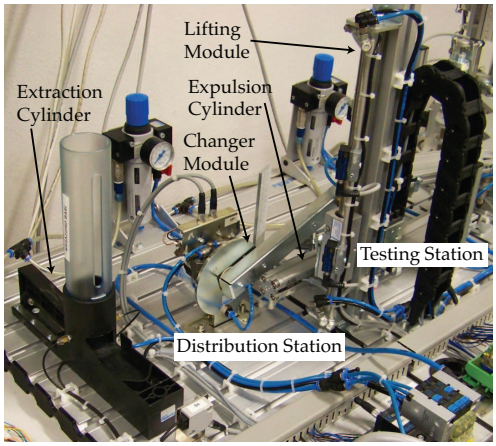


Figure 1. Distribution station and testing station of FMS

2.1 Testbed description

The testbed is part of a miniaturized flexible manufacturing system (FMS) produced by FESTO-DIDACTIC (see Fig. 1); the plant produces short-stroke cylinders, each of which are composed of a basic body, a piston, a spring and a cap. In particular, the system starts with raw pieces that are machined to produce bodies and then assembles them with the other parts to obtain the finished cylinder. In the following, the cylinders' bodies will be referred as workpieces. The FMS is composed of four stations; for the sake of brevity, only the first two stations are considered. The first station, called the distribution station, is composed of a warehouse that contains up to eight raw workpieces. During normal operation, one workpiece at a time is expelled from the warehouse by means of an Extraction Cylinder and then transferred to the second station, called the testing station, by means of a Changer Module. A Suction Cup is used to hold the workpiece during the transportation. Upon arrival of a workpiece in the testing station a sensor is used to first check its color. If the color is as required, a Lifting Module moves the workpieces to a height tester to check if the height is also acceptable. If the measurement outcome is positive, the raw piece is forwarded to the next station by means of an

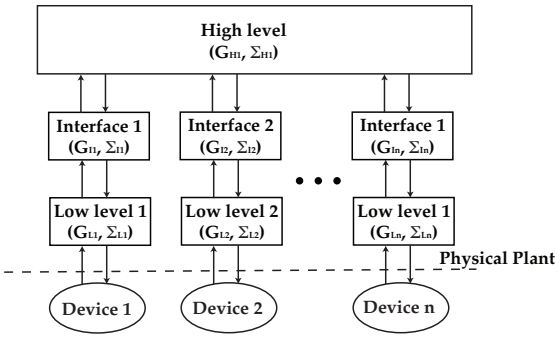


Figure 2. Hierarchical architecture

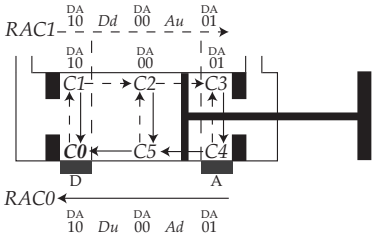


Figure 3. Illustrative example: Single-acting device

Expulsion Cylinder. Otherwise, the base is lowered by the Lifting Module and then discarded by the Expulsion Cylinder.

2.2 Hierarchical architecture

The role of the control logic in an automated system consists of proper management of all field devices so that the overall system behavior fulfils some assigned target requirements. By analyzing common classes of sensors and actuators that equip an automated manufacturing system, we can define general categories of low-level devices on the basis of the number of actuation mechanisms (single or double acting) and the number of feedback signals (double, single, or no feedback); this characterization is presented in [4]. Employing this classification, it is possible to categorize the devices that compose the individual stations: the Extraction Cylinder turns out to be a single-acting/double-feedback device, the Changer Module a double-acting/double-feedback device, the Suction Cup a single-acting/no-feedback device, the Lifting Module a double-acting/no-feedback device, and the Expulsion Cylinder a single-acting/single-feedback device. The hierarchical architecture envisioned for the control of the miniaturized FMS is depicted in Fig. 2. The proposed architecture is based on the concept of structural separation between "policies" and "actions," and is a development of the methodology to decompose the entire system into a set of basic actions that together give the entire system behavior as explained in [3] and [4]. This architecture specifically consists of three levels:
(i) *High level*: This level embeds the control policy of the system; it is devoted to the management of different operational modes and to the realization of the sequences of actions they imply. This policy is structured as sequences of activation and deactivation commands for the lower levels, but does not provide direct actuation commands.
(ii) *Interface*: This level translates high-level commands into the language of the generic low-level devices.
(iii) *Low level*: This level contains the basic control loops which implement the actuation mechanisms.

A related hierarchical architecture is proposed in [7], where the focus is on the modular verification of global properties in the context of supervisory control problems. In this architecture (called Hierarchical Interface-based Supervisory Control) a system is decomposed into one high-level subsystem and multiple low-level subsystems which communicate through well-defined interfaces. If each subsystem and its interfaces satisfy certain local conditions, then global properties can be guaranteed without constructing the full system model.

Our focus in this paper is on the modeling methodology rather than on the efficient verification of properties. We aim for generic low-level component models that are reusable, leaving to the high-level model application-dependent issues regarding "what to do" and "when to do it." Actuation mechanisms used to implement the desired high-level actions deal with "how to do it" issues, which are dependent on the low-level physical constraints. In this context, the models of the low-level components are application-independent and thereby reusable. To ensure reusability of the low-level components, we require that they have a standard structure and also a standard set of input/output commands. This is why the interface, the middle level of the architecture, is necessary. Its role is to map the high-level commands into low-level commands where the low level is linked to the actual physical devices. The low-level control logic also supports the diagnosis of certain faults on the basis of straightforward cause-effect relationships.

All the components of the architecture can be modeled as finite-state automata (indicated by the symbol G in Fig. 2) over given event sets (indicated by the symbol Σ in Fig. 2). In order to ensure the desired properties of modularity and reusability for the architecture, the following assumptions are made:

$$\Sigma_{H1} \cap \Sigma_{Ln} = \emptyset\,, \tag{1}$$
$$\Sigma_{H1} \cap \Sigma_{In} \neq \emptyset\,, \tag{2}$$
$$\Sigma_{In} \cap \Sigma_{Ln} \neq \emptyset\,, \tag{3}$$
$$\Sigma_{Li} \cap \Sigma_{Lj} = \emptyset\,. \tag{4}$$

Equations (1), (2) and (3) ensure the separation between the control policy and the actuation mechanisms, while equation (4) guarantees the modularity and reusability of the low-level control/diagnostic logic. The model-building methodology that we propose for the low level is presented in the next section.

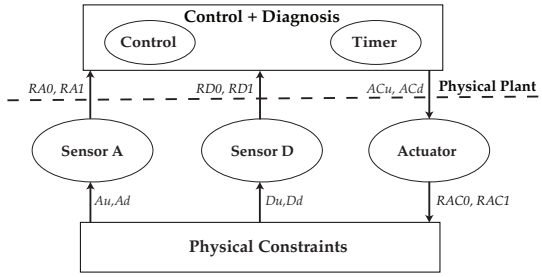
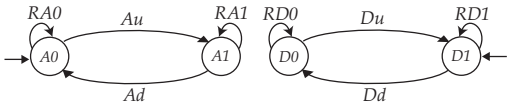
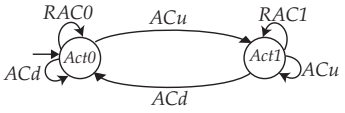


Figure 4. Physical and logical components of the low level of the architecture



(a) No fault model $G_{L,Anf}$ (b) No fault model $G_{L,Dnf}$



(c) No fault model $G_{L,Actnf}$

Figure 5. Nominal models of the sensors and actuator

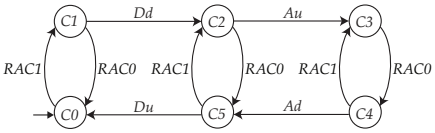


Figure 6. Physical Constraint Automaton (PCA) $G_{L,PCA}$

3 Model-Building Methodology

3.1 The low level

The architecture in Fig. 2 associates several low levels and their interfaces with the high level, according to the physical devices comprising the system. In our FMS testbed we have 6 low-level modules and associated interfaces. We describe our approach to building automata models for a single-acting device and will generalize the procedure to other classes of devices. The desired model is the composition of automata models of hardware components (actuators and sensors) and models of logic components (control logic, monitoring logic, and logic constraints as timers). Our objective is to embed in the low level not only the low-level supervisory logic, but also the diagnostic logic in order to: (i) achieve reusable software as the low-level devices are application-independent; and (ii) send to the high level the smallest amount of diagnostic information possible. To clarify these crucial points we start from the description of the Extraction Cylinder (single acting/double feedback) shown in Fig. 3. This device has two end-of-stroke sensors: sensor A signals when the device is in the activated position and sensor D signals when the device is in the deactivated position. Initially, the device is deactivated, that is, the pneumatic piston is completely retracted. The device is driven by the actuation command AC. When AC is observed to be high, the device moves from left to right unless it was already in the rightmost position (activated). Likewise, when AC is observed to be low the device moves from right to left, again unless it was already at the leftmost position (deactivated).

The model of a generic low-level component can be constructed by the interconnection of hardware models and logic models as shown in Fig. 4. We start constructing a low-level component model by first introducing nominal models for the physical components as shown in Fig. 5. Sensor A has two steady states $A0$ and $A1$; events $RA0$ and $RA1$ indicate if the sensor is read to be low or high, respectively, while events Au and Ad indicate that the sensor output changed from low to high, or high to low, respectively. The rising and falling of sensor readings (Au and Ad for example) are not observed by the controller: only on the occurrence of polling a sensor ($RA0$ and $RA1$ for example) does the controller know if the sensor has changed state. Furthermore, since the controller does not directly control the rising and falling of sensor readings, Au and Ad are uncontrollable. The controller, however, does have control over when a sensor is read. Sensor D is modeled in an analogous manner. The actuator also has two steady states $Act0$ (retracted) and $Act1$ (extended); events $RAC0$ and $RAC1$ correspond to the movement of the device in one direction or the other. From the point of view of the controller these events are unobservable because the controller cannot observe the movement of the device, it can only read sensor states. Events ACu and ACd are the commands used by the control logic to switch the position of the actuator and hence are observable and controllable. The models in Fig. 5 consider the actuator and sensors as acting alone; when they are interconnected into the device as in Fig. 3, they interact following a set of physical constraints based on the physical structure of the device (a similar idea is presented in [16]). These physical constraints can be modeled by analyzing the behavior of the device shown in Fig. 3. When the device is in the deactivated position sensor D and sensor A together read 10. After control command ACu the device starts to move towards the activated position (event $RAC1$ occurs). As the device leaves its initial position the sensors read 00 until the device reaches the activated position where the sensors read 01. During this movement the device can be in three different states (namely $C1$, $C2$ and $C3$) reflected by the different readings of the sensors. The opposite movement generates states $C4$, $C5$ and $C0$ in which the sensors in the device have the same respective readings as in $C3$, $C2$ and $C1$. Figure 6 shows the *Physical Constraint Automa-*

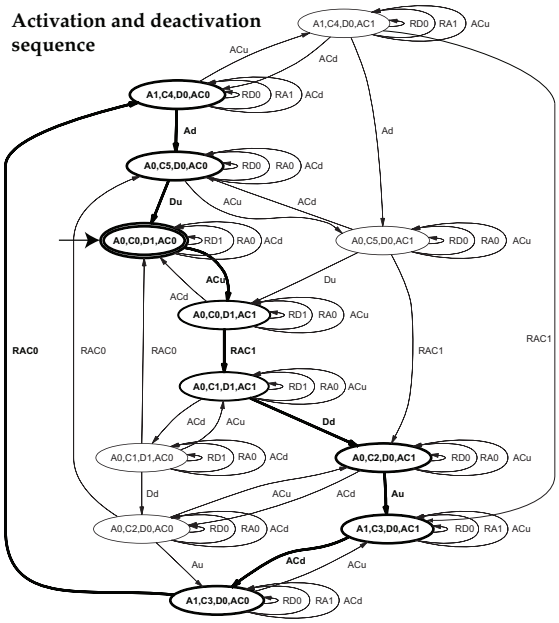
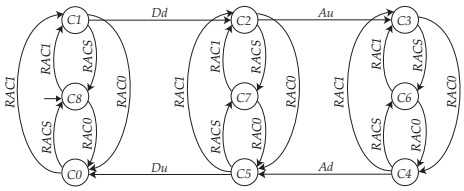
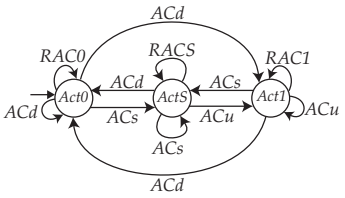


Figure 7. Composition of nominal sensors, actuator and PCA: $G_{L,CompNom}$

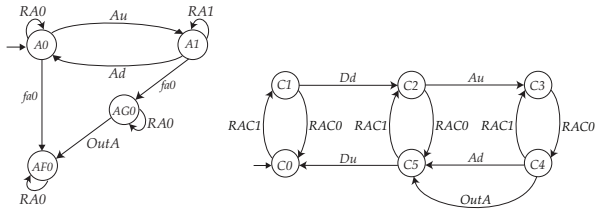


(a) Physical Constraint Automaton for a double-acting device (PCA_DA): G_{L,PCA_DA}



(b) No fault actuator model, $G_{L,Actnf\_DA}$

Figure 8. Components for a double-acting device



(a) Faulty model of sensor A, $G_{L,Afa0}$ (b) Physical Constraint Automaton with fault $fa0$, G_{L,PCA_A}

Figure 9. Faulty models

ton (PCA) that models these constraints. Composing the automata in Fig. 5 and Fig. 6 we obtain the automaton in Fig. 7.

To summarize, sensors A and D and the actuator (called Act for brevity) respectively have the following observable event sets $\Sigma_{A,o} = \{RA0, RA1\}$, $\Sigma_{D,o} = \{RD0, RD1\}$, $\Sigma_{Act,o} = \{ACu, ACd\}$ and the following unobservable event sets $\Sigma_{A,uo} = \{Au, Ad\}$, $\Sigma_{D,uo} = \{Du, Dd\}$ and $\Sigma_{Act,uo} = \{RAC0, RAC1\}$. Furthermore, each of the observable events are controllable and each of the unobservable events are uncontrollable. The automaton shown in Fig. 7 is the model of a single-acting/double-feedback device.

The Changer Module of the FESTO system is a double-acting/double-feedback device; to build this model we do not need to build a completely new model. Only changes to the PCA automaton and the actuator model are necessary. This device can hold its current position in the middle of the movement, whereas a single-acting device can only hold its position when the device is in deactivated or activated position. Figure 8(a) shows the PCA automaton where the new unobservable event $RACS$ models the hold position event, while Fig. 8(b) depicts the actuator model where the new observable event ACs models the hold position command. With similar considerations it is possible to build a model for any type of device, composing one or two sensors, the actuator and PCA automaton to model a single- or double-feedback device. For a complete list of models, the interested reader is referred to [14].

3.2 Modeling faults at the low level

Generally speaking, embedding faults in device models such as the one in Fig. 7 is a difficult task if one works with the entire automaton modeling the device. Fortunately, following the modular approach of the preceding section, modeling a fault in a physical component can be accomplished by simply modifying the underlying physical models and the constraint model according to the local effect of the fault. Consider for example the case in which the activation sensor A can be stuck at its low level; we model this fault with the unobservable event $fa0$. When sensor A fails in this manner the sensor cannot read high anymore. This fault has two consequences, a *local* consequence on the sensor modeled by the automaton in Fig. 9(a) getting stuck in state $AF0$, and a *global* consequence on the whole device modeled by the PCA in Fig. 9(b). The global consequence is that the device can move from state $C4$ to state $C5$ even without the occurrence of event Ad which represents the nominal falling edge of the signal output from sensor A. The new unobservable and uncontrollable event $OutA$ is used to model this situation in the sensor automaton and in the PCA. Figure 10 shows the result of the parallel composition of the

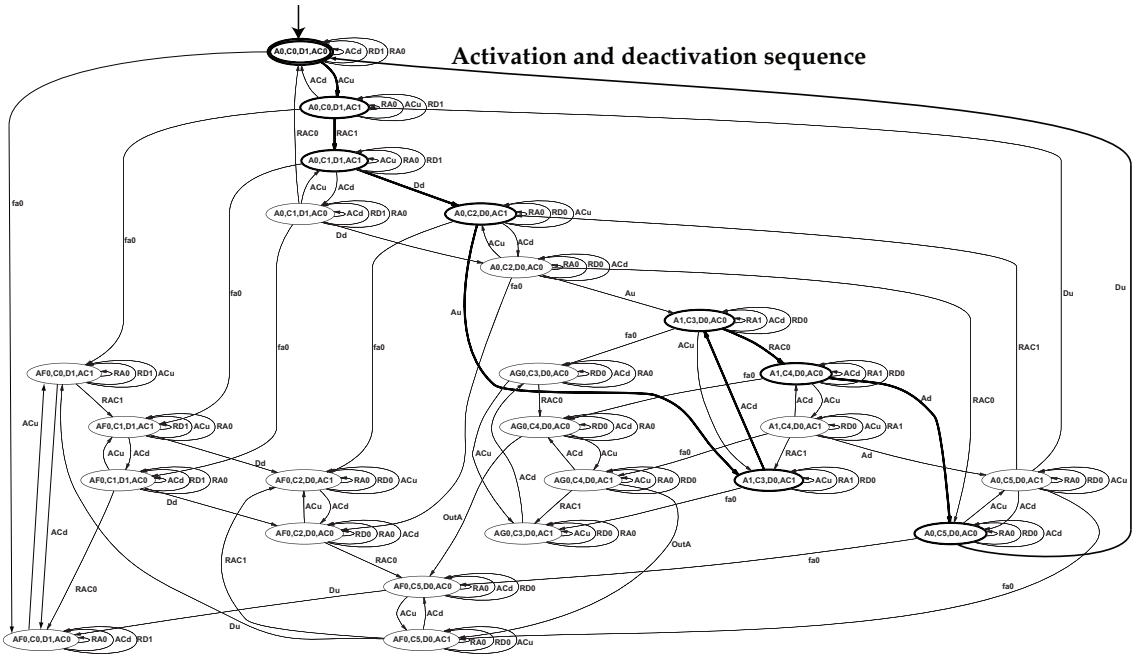


Figure 10. Nominal and faulty model for the single-acting device, $G_{L,Compfa0}$

nominal automata $G_{L,Dnf}, G_{L,Actnf}$ of Fig. 5 and the faulty automa $G_{L,Afa0}, G_{L,PCA\_A}$ of Fig. 9. Note that the resulting automaton contains both the nominal and faulty behavior of the device and is, therefore, suitable to be used in model-based diagnostics algorithms like those employed in [13] and [2]. It is important to remark that since the model in Fig. 10 is completely case-independent, the monitoring code designed from it is reusable. With further simple modifications to the automata in Fig. 5 and Fig. 6, it is also possible to model scenarios including multiple faults; for the sake of brevity, such models are not reported here, but the reader can find them in [14].

Remark 1 The automata models of Fig. 7 and Fig. 10 do not include any control logic, but the nominal activation and deactivation sequence of the device that could be implemented by a controller is shown in bold. Of course, the automata can perform other sequences depending on the actual control logic employed, or in the case of Fig. 10, due to the occurrence of faults.

Remark 2 The operation of the device is characterized by three sources of information, readings from the two sensors and commands to the actuator. This information indicates the state of the device. The state of the device can be changed by a new control command (events ACu, ACd) that forces a change in the states of the sensors (events Au, Ad, Du, Dd) through the constraint automaton PCA. The self-loops of the automata in Fig. 7 and Fig. 10 can be thought of as "outputs" emitted from a given state in much the same manner as a Moore automaton.

Remark 3 In our approach to decomposing the control logic into two levels, we attempt to model faults in a single low-level component where possible. This approach, when it can be done, has the advantage that the diagnostic logic can be designed and implemented in the different low-level components independently, without consideration of the high-level.

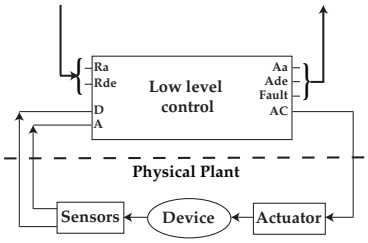


Figure 11. Single actuator device control

4 CONTROL AND MONITORING OF LOW-LEVEL DEVICES

4.1 Supervisory control

As explained in Section 2.2 the high-level supervisor sends events to the low level in order to force basic actions as depicted in Fig. 11. More specifically, the event Ra is used to request an activation of the device, while the event Rde is employed to request a deactivation of the device. Note that we use two request events; in this way the policy is independent from the implementation of the device, which is hidden in the low-level control logic. In this manner the same control logic can be used in a single-acting device or a double-acting device. When the device has accomplished the high-level request it notifies the high level using event Aa (activation accomplished) and event Ade (deactivation accomplished). The desired behavior for the controlled device is depicted by the automaton

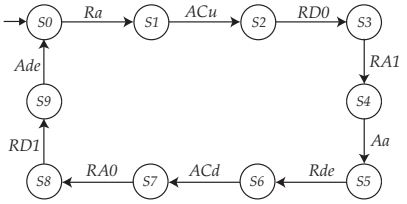


Figure 12. Desired behavior automaton $E_{L,ConNom}$ for low-level control of a single-acting device

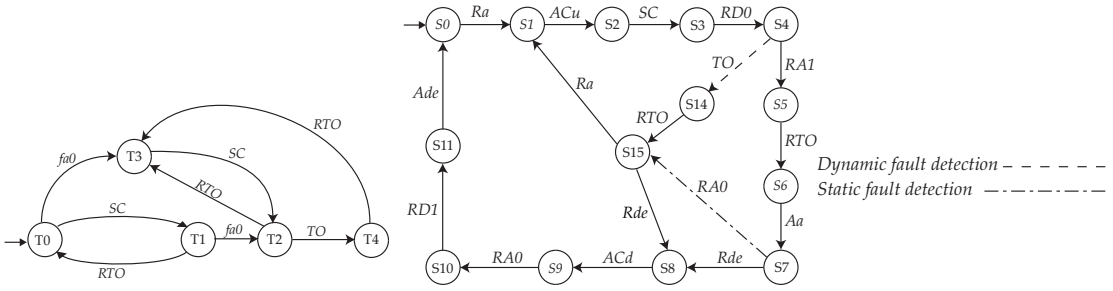
$E_{L,ConNom}$ shown in Fig. 12. Referring to Fig. 12 where the device is initially inactive, it is desired that when the high level asks for an activation by sending the event Ra, it causes the command ACu to be sent to the actuator by the low level. After a given amount of time, event $RD0$ then will signal that the device is not inactive anymore. After a further amount of time, event $RA1$ will indicate that the device is activated. It is desired that this fact cause event Aa to be sent from the low level to the high level to acknowledge the accomplishment of the original request. At this point, the low-level control will wait for a deactivation request (event Rde) and a similar process will be carried out as with the activation request. The high-level request and answer events $\{Ra, Aa, Rde, Ade\}$ are observable and controllable. Composing the desired behavior in Fig. 12 with the nominal sensors, actuator and PCA models of Fig. 5 and Fig. 6, we obtain the controlled device model $G_{L,DevNom}$ which has 18 states and 20 transitions. Using formal discrete event theory, it can be shown that the desired behavior of Fig. 12 is achievable by supervisory controller (see [11]) since it is both controllable and observable. Moreover, it turns out that the automaton in Fig. 12 can be used as realization of the corresponding supervisor.

4.2 Dealing with device faults

If we consider the models for the single-acting device with fault $fa0$ (see Fig. 9) composed with the nominal controller in Fig. 12, we obtain the controlled device model $G_{L,Devfa0}$ from the parallel composition of models $G_{L,Afa0}, G_{L,Dnf}, G_{L,Actnf}, G_{L,PCA\_A}$ and $E_{L,ConNom}$. The automaton $G_{L,Devfa0}$ has 38 states and 63 transitions. Since sensor A cannot rise to true, the desired sequence $E_{L,ConNom}$ is stuck in state $S3$ and it is easy to understand that this condition generates a deadlock in $G_{L,Devfa0}$. To avoid this deadlock, we consider a new logical model that considers also timing constraints. The aim of this strategy is twofold: (i) embed basic fault diagnosis in the low-level control logic, leaving to the high level only the diagnostic task of detecting complex faults that involve multiple devices; and (ii) avoid deadlock due to faults. In order to avoid deadlock in the controlled faulty device, we substitute the desired behavior $E_{L,ConNom}$ shown in Fig. 12 with the supervisor $E_{L,ConDiag}$ depicted in Fig. 13(b). $E_{L,ConDiag}$ is generated by designer understanding of the system supposing that the time needed to activate or deactivate the device is bounded by some known amount when the device is operating correctly. The diagnostic logic will check the consistency of this rule during the evolution of the device to determine whether or not a fault has occurred. Such a diagnostic algorithm will not require the introduction of timed automata as the necessary timing information can be captured by a timeout event TO that signals the violation of the deadline. With this in mind, we consider the timer model in Fig. 13(a). Event SC is used to start the timer, while event RTO is used to reset the timer. Note that the timeout event TO can only occur after the occurrence of the fault $fa0$ because this is supposed to be the only case in which the temporal rule is violated. Note that SC, RTO and TO are observable and controllable events. The new supervisor embeds not only control like $E_{L,ConNom}$ did, but also fault detection. If the device is in the activated position waiting for a request of deactivation (state $S7$) and the sensor A spontaneously changes its output to low, the fault $fa0$ has occurred and it is detected when event $RA0$ occurs taking the supervisor to state $S13$. When a fault is detected without a movement of the system we classify this as *static fault detection*, shown by point-dashed lines in Fig. 13(b). If the fault $fa0$ occurs when the sensor value of A is low, the fault can only be detected when a movement of the device occurs which should force the value A to high. For this reason, when the supervisor receives a request of activation (event Ra), this causes not only the event ACu but also the event SC that starts the timer. In this case the device never reaches the activated position because the event $RA1$ never occurs and so the control does not reset the timer (event RTO). The fault is then detected in state $S12$ by the event TO that signals the violation of the time deadline. We classify this as *dynamic fault detection* indicated by dashed lines in Fig. 13(b). Composing the new low-level supervisor $E_{L,ConDiag}$ in Fig. 13(b) and the timer model $G_{L,Tfa0}$ in Fig. 13(a) with the sensors $G_{L,Afa0}, G_{L,Dnf}$, actuator $G_{L,Actnf}$ and PCA model G_{L,PCA_A}, we obtain the controlled device model $G_{L,Totfa0}$; the resulting automaton has 57 states and 94 transitions. It is possible to formally check the diagnosability of the controlled device using the diagnoser theory for discrete event systems modeled by automata [13]. The resulting analysis demonstrates that the fault can be identified under all conditions and further that there is no deadlock. The reader can find the details of this analysis in [14].

5 CONCLUSION

We presented a general approach to discrete event modeling of physical behavior and control logic in industrial automation. The key features of the proposed approach are its modularity, exploiting parallel composition to obtain the complete system model from models of individ-



(a) Timer model for fault $fa0$, $G_{L,Tfa0}$ (b) Supervisor $E_{L,ConDiag}$ for the single acting device considering fault $fa0$

Figure 13. New components for fault detection

ual components, and the reusability of the generic component models. The reusability of component models is made possible by the construction of a so-called "physical constraint automaton" that captures the physical coupling of generic components in a given automated system. In our general approach, we first build fault-free models then show how to extend them to include faulty behavior, preserving modularity. We also employ a hierarchical decomposition that separates the control logic into a high level that manages the sequence of control actions and a low level that implements the control actions. These two levels are coupled through an interface. Our models have the property that sensor readings at each physical state are represented as event self-loops; this achieves the same objectives as Moore automata, but in an entirely event-based framework. In this context, we are able to leverage the formal techniques from supervisory control and fault diagnosis for discrete event systems modeled as automata. In particular, diagnosability analysis can be performed on the models that include faulty behavior. One of the goals is to ensure that component faults can be diagnosed at the lower level of the hierarchy.

References

[1] M. Bonfé and C. Fantuzzi. Application of object-oriented modeling tools to design the logic control system of a packaging machine. *IEEE International Conference on Control application Industrial Informatics*, pages 569–574, 2004.

[2] E. Dumitrescu, A. Girault, H. Marchand, and E.Rutten. Optimal discrete controller synthesis for modeling fault-tolerant distributed systems. *Proceeding of 1st IFAC workshop on Dependable Control of Discrete Systems*, 2007.

[3] E. Faldella, A. Paoli, M. Sartini, and A. Tilli. Hierarchical control architectures in industrial automation: a design approach based on the generalized actuator concept. *Proceedings of 17th IFAC WC*, 2008.

[4] E. Faldella, A. Paoli, A. Tilli, M. Sartini, and D. Guidi. Architectural design patterns for logic control of manufacturing systems: the generalized device. *XXII International Symposium on Information, Communication and Automation Technologies*, 2009.

[5] L. Ferrarini, R. Brusa, and C. Veber. A pragmatic approach to fault diagnosis in hydraulic circuits for automated machining: a case study. *4th IEEE Conference on Automation Science and Engineering*, 2008.

[6] A. Hellgren, B. Lennartson, and M. Fabian. Modelling and plc-based implementation of modular supervisory control discrete event systems. *Proceedings of 6th Int. Workshop on Discrete Event Systems*, 2002.

[7] R. Leduc, M. Lawford, and P. Dai. Hierarchical interface-based supervisory control of a flexible manufacturing system. *IEEE Transactions on Control Systems Technology*, 14(4):654 –668, 2006.

[8] J. Lunze. Fault diagnosis of discretely controlled continuos systems by means of discrete-event models. *Discrete Event Dynamic Systems Theory and Applications*, 18(3):385 –413, 2008.

[9] J. M. Machado, B. Denis, J. J. Lesage, J. Faure, and C. L. F. da Silva. Logic controllers dependability verification using a plant model. *Proceedings of 3th IFAC Workshop on Discrete-Event System Design*, 2006.

[10] A. Paoli, M. Sartini, and S. Lafortune. A fault tolerant architecture for supervisory control of discrete event systems. *Proceedings of the 17th IFAC World Congress*, 2008.

[11] P. J. G. Ramadge and W. M.Wonham. The control of discrete event systems. *in Proc. IEEE*, 77(1):81 –98, 1989.

[12] J. Roussel and A. Giua. Designing dependable logic controllers using the supervisory control theory. *Proceedings of the 16th IFAC World Congress*, 2005.

[13] M. Sampath, R. Sangupta, and S. Lafortune. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40(9):1555 – 1575, 1995.

[14] M. Sartini, A. Paoli, R. C. Hill, and S. Lafortune. Model-building for automated manufacturing systems: a discrete event systems approach, casy technical report, avaible from the authors. 2010.

[15] M. Sayed-Mouchaweh, A. Philippot, V. Carre-Menetrier, and B. Riera. Fault diagnosis of discrete event systems using components fault-free models. *Proceedings of the 20th International Workshop on Principles of Diagnosis*, 2009.

[16] A. Tilli and A. Paoli. Rule-based composable modelling of industrial automation automata under nominal and faulty conditions. *Proceedings of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes*, 2009.

[17] V. Vyatkin and H. M. Hanisch. Formal modeling and verification in the software engineering framework of IEC 61499: a way to self-verifying systems. *IEEE International Conference on Emerging Technologies and Factory Automation*, pages 113–118, 2001.