# Fault-Tolerant Control of Discrete Event Systems based on Fault-Accommodating Models

**Thomas Wittmann** [*] **Jan Richter** [**] **Thomas Moor** [*]

[*] *Lehrstuhl für Regelungstechnik, Friedrich-Alexander Universität Erlangen-Nürnberg, Cauerstraße 7, 91058 Erlangen, lrt@rt.eei.uni-erlangen.de*
[**] *Siemens AG, Industry Sector, Gleiwitzer Str. 555, 90475 Nuremberg*

**Abstract:** Fault-tolerant control systems with discrete-event dynamics allow for differing sets of design requirements, that specify the system's behaviour during nominal operation and in the case of component degradation or component malfunction. This paper is concerned with the design of fault-tolerant control algorithms for discrete event systems in the framework of supervisory control theory. Its main contribution is a modelling framework that describes the evolution of systems which are potentially subject to faults. These models are called *fault-accommodating models*. Within this context, the occurrence of faults is modelled by means of unobservable and uncontrollable events. Hence, the design problem of fault-tolerant controllers is transformed to a supervisory control problem under partial observation. Consequently, there is no need for explicitly diagnosing the occurrence of faults and relaxed diagnosability conditions are applied. Finally, the paper provides extensive examples in order to illustrate the application of all derived methodological results. All computations needed for controller design and system analysis were implemented using a freely available software toolbox.

Keywords: discrete-event systems, supervisory control, fault-tolerant systems, applications and faul-tolerant control

## 1. INTRODUCTION

A discrete event system is called fault-tolerant if it satisfies suitable, but not necessarily identical specifications both before and after the occurrence of faults. This paper is concerned with the design of fault-tolerant control algorithms in the framework of supervisory control theory. Therein, it is possible to formally analyse the plant dynamics as well as the dynamics of the closed loop system. Especially the guaranteed fulfilment of a given safety specification is of special interest w.r.t. the design of fault-tolerant control systems.

The main contribution of this paper is a modelling framework for systems subject to spontaneously occurring faults, that allows for the transformation of the fault-tolerant control problem to a standard supervisory control problem. We introduce a model class that makes the logical order of nominal events and fault events, together with information on the fault's impact, available for controller design. Solving a standard problem from supervisory control theory supersedes the need for a diagnoser as well as the need for a switching mechanism, that handles the changeover from the nominal control policy to the control policy in the presence of faults. Furthermore, well-known algorithms can be used to solve the fault-tolerant control problem at hand.

An approach to fault-tolerant supervisory control based on the concept of stability and language-convergence is presented in [Wen et al., 2008]. Therein, a system is called fault-tolerant, if its behaviour converges towards the nominal behaviour in a finite number of steps. Hence, only faults whose impact is reversible after a finite number of events are considered. Faults can be interpreted as model uncertainties, thus methods from robust control (see Bourdon et al. [2005], Takai [2000], Cury and Krogh [1999], Lin [1993]) can be used to compute fault-tolerant controllers. In the context of discrete event systems, uncertainties are respected by a family of possible plant models. Each member of this family that represents a defective plant behaviour, models the impact of the fault at hand, but not its previous history. That is, spontaneously occurring faults are not considered. In [Park and Lim, 1999] an approach to robust control is extended by dividing strings into either tolerable or non-tolerable strings.

In [Paoli et al., 2008] the fault-tolerant control design problem for discrete event systems is solved using a three-step algorithm. It includes the detection of faults before the nominal design requirements are violated, stopping the system after the diagnosis of a fault and enforcing a different specification on the defective system. The existence of the proposed algorithm is subject to diagnosability conditions based on the results reported in [Sampath et al., 1995] and controllability conditions that claim a possible controller intervention, before the system exhibits some forbidden behaviour. Its implementation is based on a diagnoser and a set of pre-computed controllers. After the diagnosis of a fault, the nominal supervisor is disabled and replaced by a supervisor that was designed according to the fault at hand. Besides the aforementioned restrictions, an additional switching mechanism is needed to choose a supervisor according to the diagnosed fault and to synchronise the chosen supervisor with the plant's current state. Paoli's approach is settled in the framework presented in [Blanke et al., 2010] and categorised as an active approach to fault-tolerant control.

According to the classification as stated in [Blanke et al., 2010] the approach presented in this paper is an active approach, since the behaviour of the closed-loop system changes according to the occurred fault. Note that the concept of fault accommodation proposed in [Blanke et al., 2010] is not related to the approach presented in this paper. The structure of fault-accommodating models is closely related to Paoli's approach. However, an explicit fault-diagnosis or synchronisation mechanism is avoided. Example 6 shows that restrictions, imposed by the use of a diagnoser or switching conditions are considerably relaxed. Similar to approaches from robust control, the fault impacts are modelled separately. Whereas in robust control each fault impact is covered by a separate model, the presented approach differentiates only the nominal behaviour and the behaviour of the defective plant. Furthermore, fault-accommodating models account for spontaneously occurring faults. In contrast to approaches based on state stability and language convergence, in this paper the defective plant is not required to eventually comply with the nominal plant behaviour.

The main part of this article is organised as follows. Section 2 provides some basic notations around formal languages and automata. In addition, selected basic concepts from supervisory control are recalled. Thereafter, the concept of fault-tolerant control of discrete event systems based on fault-accommodating models is introduced in Section 3. Section 4 provides an application example from the field of manufacturing automation. Finally, some concluding remarks are stated.

## 2. PRELIMINARIES

This section recalls basic notations and concepts from formal languages and supervisory control theory, see [Ramadge and Wonham, 1987, 1989, Wonham, 2009].

### *2.1 Formal Languages and Automata*

An alphabet $\Sigma$ is a finite set of events $\sigma$. Let $\Sigma^*$ denote the Kleene closure over $\Sigma$. Elements $s \in \Sigma^*$ are termed *strings*. By definition $\Sigma^*$ includes the *empty string* $\varepsilon$. Given strings $r, s, t$, the *concatenation* of $r$ and $t$ is denoted by $rt$. Any string $r$ such that there exists a $t \in \Sigma^*$ with $rt = s$ is called a *prefix* of $s$.

A *formal language* $L \subseteq \Sigma^*$ is a subset of $\Sigma^*$. Its *prefix-closure* $\overline{L} = \{ r \in \Sigma^* \mid (\exists t \in \Sigma^*)[rt \in L]\}$ contains all sequences, which are prefixes of strings in $L$. A formal language $L$ with $L = \overline{L}$ is called *prefix-closed*. Given formal languages $L$ and $K$, the set $L - K := \{ s \in L \mid (s \notin K)\}$ denotes the *language-difference*. A language $L \in \Sigma^*$ is called *regular* if it is built upon *regular expressions*. Every regular language is recognised by a *finite automaton* and vice versa.

A *finite automaton* $G$ is a tuple $(Q, \Sigma, \delta, q_0, Q_\mathrm{m})$, with the state set $Q$, the alphabet $\Sigma$, the transition function $\delta : (Q \times \Sigma) \to Q$, the initial state $q_0$ and the set of marked states $Q_\mathrm{m} \subseteq Q$. Note that $\delta$ is a partial function. In addition the transition function can recursively be extended to strings. Therefore, let $\delta(q_0, \varepsilon) = q_0$ denote the beginning of the recursion. For any string $s \in L(G), \sigma \in \Sigma$ set $\delta(q_0, s\sigma) := \delta(\delta(q_0, s), \sigma)$. An automaton's *generated language* is defined according to $L(G) = \{s \in \Sigma^* \mid (\exists q \in Q)[\delta(q_0, s) = q]$ and $L_\mathrm{m}(G) := \{s \in \Sigma^* \mid (\exists q \in Q_\mathrm{m})[\delta(q_0, s) = q]$ denotes the automaton's *marked language*. The *dynamics* or *behaviour* of discrete-event systems modelled by a finite automaton is given by means of the marked language.

### *2.2 Supervisory Control*

Given is an alphabet $\Sigma$, partitioned according to the *controllability* and *observabiltity* attributes of the contained events, hence $\Sigma = \Sigma_\mathrm{c} \dot\cup \Sigma_\mathrm{uc}$ and $\Sigma = \Sigma_\mathrm{o} \dot\cup \Sigma_\mathrm{uo}$. Furthermore $\Sigma_\mathrm{c} \subseteq \Sigma_\mathrm{o}$ is required to hold.

The *natural projection* $\mathrm{p_o} : \Sigma^* \to \Sigma_\mathrm{o}^*, \Sigma_\mathrm{o} \subseteq \Sigma$ is recursively defined. Let $\mathrm{p_o}(\sigma) = \varepsilon$ if $\sigma \notin \Sigma_\mathrm{o}$ and $\mathrm{p_o}(\sigma) = \sigma$ otherwise, then define $\mathrm{p_o}(\varepsilon) = \varepsilon$ and note that $\mathrm{p_o}(s\sigma) = \mathrm{p_o}(s)\mathrm{p_o}(\sigma)$ holds for an arbitrary string $s \in \Sigma^*$. The *inverse natural projection* is given according to $\mathrm{p_o}^{-1} : \Sigma_\mathrm{o}^* \to 2^{\Sigma^*}, \mathrm{p_o}^{-1}(s) := \{t \in \Sigma^* \mid \mathrm{p_o}(t) = s\}$. The natural projection can be extended to languages, in compliance to $\mathrm{p_o} : 2^{\Sigma^*} \to 2^{\Sigma_\mathrm{o}^*}, \mathrm{p_o}(L) = \{ r \in \Sigma_\mathrm{o}^* \mid (\exists s \in L)[\mathrm{p_o}(s) = r]\}$ and $\mathrm{p_o}^{-1} : 2^{\Sigma_\mathrm{o}^*} \to 2^{\Sigma^*}, \mathrm{p_o}^{-1}(L) = \{ s \in \Sigma^* \mid \mathrm{p_o}(s) \in L\}$.

A *supervisor*, is a map $S : \overline{L} \to \Gamma := \{\gamma \in \Sigma \mid \Sigma_\mathrm{uc} \subseteq \gamma\}$, where $\Gamma$ is the set of all *control patterns*. For brevity $S/L$ denotes the set of all strings contained in $L$, that are compatible with $S$. A supervisor is called *nonblocking* w.r.t. $L$ if $S/\overline{L} = \overline{S/L}$, *feasible* if $(\forall s, s' \in L)[\mathrm{p_o}s = \mathrm{p_o}s' \Rightarrow S(s) = S(s')]$, and *admissible* if $S/L \subseteq L$ and $(\forall s, s')[s \in S/L \& s' \in L \cap \overline{S/L} \& \mathrm{p_o}s' = \mathrm{p_o}s \Rightarrow s' \in S/L]$.

A nonblocking, feasible and admissible supervisor disables only controllable events, does not prevent the plant from reaching its markings and the occurrence of unobservable events does not influence the supervisor's decision on which events are permissible and which are not. Furthermore two under observation equal strings, are accepted if they are both marked w.r.t. the plant.

Given two languages $L, K \in \Sigma^*$ and let $\mathrm{p_o}$ denote the natural projection of $\Sigma$ onto $\Sigma_\mathrm{o}$, then $K$ is *relatively closed* w.r.t. to $L$ if $K = \overline{K} \cap L$, $K$ is *controllable* w.r.t. $L$ and $\Sigma_\mathrm{uc}$ if $\overline{K}\Sigma_\mathrm{uc} \cap \overline{L} \subseteq \overline{K}$ and $K$ is *normal* w.r.t. $L$ and $\mathrm{p_o}$ if $K = L \cap \mathrm{p_o}^{-1}(\mathrm{p_o}K)$.

In [Ramadge and Wonham, 1989] it is shown that there exists a nonblocking, feasible and admissible supervisor $S$ for $L$ such that $S/L = K$ if $K$ is controllable w.r.t. $L$ and $\Sigma_\mathrm{uc}$, $K$ is relatively closed w.r.t. $L$ and $\overline{K}$ is normal w.r.t. $\overline{L}$ and $\mathrm{p_o}$.

It can be shown that the set of all controllable, normal and relatively closed sublanguages exposes a unique supremal element, which serves as a basis for the realisation of a minimally restrictive supervisor.

## 3. FAULT-TOLERANT CONTROL BASED ON FAULT-ACCOMMODATING MODELS

This section introduces faults as uncontrollable and unobservable events. A fault-accommodating behaviour is formally defined as a language that accounts for the nominal behaviour and the defective behaviour. The nominal behaviour and the defective behaviour need to be consistent with regard to the past of an occurred fault. Hence criteria for a proper changeover from the nominal behaviour to the defective behaviour are presented. Finally, the fault-tolerant control problem is stated and reduced to a supervisory control problem under partial observation.

### *3.1 Fault Concept*

This paper focuses on *spontaneously occurring*, *permanent faults* (see Blanke et al. [2010]), hence there exists an instant of time at which the fault occurs and the fault is not reversible. Faults can be neither triggered nor detected by the controller. Therefore, the occurrence of faults is modelled by uncontrollable and unobservable events. Throughout the paper, we assume that a fault occurs only once.

Given the *nominal alphabet* $\Sigma_N$, let $F \notin \Sigma_N$ denote a distinguishable fault event and write $\Sigma = \Sigma_N \dot{\cup} \{F\}$ for the *overall alphabet* $\Sigma$. Note that, regarding the common partition of $\Sigma$ according to the events' controllability attributes and observability attributes, the fault $F$ is considered neither controllable nor observable, that is $F \in \Sigma_{uc} \cap \Sigma_{uo}$.

### *3.2 Fault-Accommodating Models*

Fault-accommodating models are intended to formalise the logical order of nominal events and faults. They are based on the *nominal behaviour* $L_N \subseteq \Sigma_N^*$ and the *defective behaviour* $L_D \subseteq \Sigma^*$ of a given system. The nominal behaviour represents the dynamical laws that govern the system in fault-free operation, whereas the defective behaviour covers a fault's previous history and its consequences for the system's subsequent behaviour. A *proper* fault-accommodating model includes only faults whose previous history complies with the nominal system dynamics.

*Definition 1.* Let $L_N \subseteq \Sigma_N^*$ denote a nominal behaviour and $L_D \subseteq \Sigma^*$ a defective behaviour, respectively, where $\Sigma = \Sigma_N \cup \{F\}$ and $F$ is a fault. The *fault-accommodating model* is a pair $(L_N, L_D)$. The fault-accommodating model $(L_N, L_D)$ is *proper* if

$$\overline{L_D} \cap \Sigma_N^* \subseteq \overline{L_N} \tag{1}$$

and

$$L_D \cap \Sigma_N^* \subseteq L_N. \tag{2}$$

□

A *fault-accommodating behaviour* $L_{FA} \subseteq \Sigma^*$ is required to account for the nominal behaviour and the defective behaviour, therefore, a natural candidate is the conjunction

$$L_{FA} = L_N \cup L_D. \tag{3}$$

Provided that a fault-accommodating plant model $(L_N, L_D)$ is proper, $L_{FA}$ can be interpreted as the nominal behaviour with optional changeover to the defective behaviour after the occurrence of the fault $F$.

*Proposition 2.* Let $(L_N, L_D)$ denote a proper fault-accommodating model, then

$$L_{FA} = L_N \cup (\overline{L_N} F \Sigma^* \cap L_D). \tag{4}$$

□

**Proof.** We first derive

$$L_D \subseteq \overline{L_N} F \Sigma^* \cup L_N \tag{5}$$

from Def. 1, Eqs. (1) and (2). Pick any $s \in L_D$. Case that $s$ consists of nominal events only, that is $s \in \Sigma_N^*$, Eq. (2) implies $s \in L_N$. Case that $s$ includes a fault event, that is $s \notin \Sigma_N^*$, we decompose $s$ such that $s = rFt$, with $r \in \Sigma_N$ and $t \in \Sigma^*$. In particular, $s \in L_D$ implies $r \in \overline{L_D}$ and by Eq. (1) $r \in \overline{L_N}$. Thus $s = rFt \in \overline{L_N} F \Sigma^* \cup L_N$. This completes the proof of Eq. (5) and we obtain Eq. (4) as follows:

$$\begin{aligned} &L_{FA} = L_N \cup L_D = \\ &L_N \cup ((\overline{L_N} F \Sigma^* \cup L_N) \cap L_D) = \\ &L_N \cup (\overline{L_N} F \Sigma^* \cap L_D). \end{aligned}$$

■

The following example illustrates the construction of a fault-accommodating plant behaviour.

*Example 3.* Simple machine - modelling

A simple machine is capable of producing two different products A and B. The subordinate process is subject to faults. In *faulty operation*, starting the production cycle A results in a product B or has no effect at all. All relevant events are summarised in Table 3.

Tab 1. Physical meaning of events
C: controllable, O: observable

| Event | Interpretation | Attributes |
|---|---|---|
| a | start process A | C, O |
| b | start process B | C, O |
| A | process A successfully finished | C, O |
| B | process B successfully finished | O |
| $\alpha$ | start subordinate process | O |
| $\beta$ | subordinate process finished | O |

Figure 1 shows a model of the nominal behaviour, whereas Fig. 2 shows the defective behaviour.

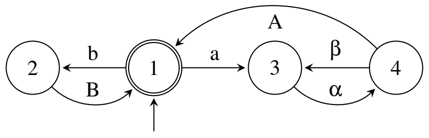


Fig. 1. Nominal behaviour

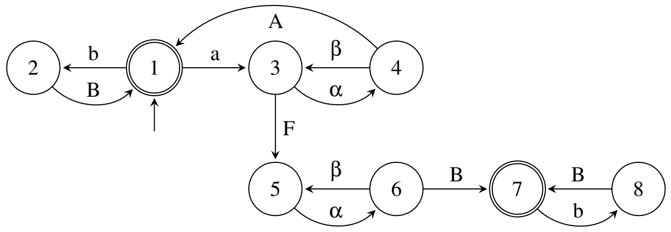


Fig. 2. Defective behaviour

Since the fault-accommodating model $(L_N, L_D)$ is proper and $L_N \subseteq L_D$, the fault-accommodating behaviour $L_{FA}$, computed according to Eq. (4), is given by the defective behaviour. □

A system which is subject to multiple faults that do not depend on each other can be modelled by the union of all related fault-accommodating behaviours. The system's malfunctions can be regarded as isolated incidents and, hence, from the view of the modeller, can be distinguished from each other. However, fault labels are rather a modelling tool to set up the defective behaviour of the plant, than a differing criteria. Hence, only a single fault event is needed to indicate that the nominal plant model does not apply any more.

### *3.3 Fault-Tolerant Control*

The occurrence of a fault is not observable, and hence the information whether the system complies with its nominal dynamical laws or not cannot be explicitly incorporated in control decisions. Since fault-accommodating models provide information on the logical order of nominal events and faults and information on the fault impact, it is sufficient to distinguish between strings that include a fault and those that consist of nominal events only. In particular, an exact characterisation of the fault using a diagnoser is not needed to ensure a desired behaviour during faulty operation. Instead, the specification requirements can be formalised using fault-accommodating models.

Consider a plant, whose dynamics is represented via the fault-accommodating model $(L_{\rm N}, L_{\rm D})$, where $L_{\rm N} \subseteq \Sigma_{\rm N}^*$ accounts for the plants nominal behaviour and $L_{\rm D} \subseteq \Sigma^*$ models the plant's defective behaviour.

The *nominal acceptable behaviour* $E_{\rm N} \subseteq \Sigma_{\rm N}^*$ restricts the nominal plant's behaviour and the *defective acceptable behaviour* $E_{\rm D}$ narrows the plant's operation in the case of faults. Given a proper fault-accommodating model $(E_{\rm N}, E_{\rm D})$, the *acceptable behaviour* $E_{\rm A} \subseteq \Sigma^*$ can be written as

$$E_{\rm A} = E_{\rm N} \cup (\overline{E_{\rm N}} F \Sigma^* \cap E_{\rm D}), \tag{6}$$

according to Prop. 2.

A supervisor that restricts the plant's fault-accommodating behaviour to the acceptable plant behaviour is called *fault-tolerant*.

*Definition 4.* Consider the fault-accommodating models $(L_{\rm N}, L_{\rm D})$ and $(E_{\rm N}, E_{\rm D})$, where $(L_{\rm N}, L_{\rm D})$ models the plant behaviour and $(E_{\rm N}, E_{\rm D})$ represents the design requirements. Further, $L_{\rm FA}$ and $E_{\rm A}$ denote the fault-accommodating behaviour according to Eq. (4) associated with the fault-accommodating models $(L_{\rm N}, L_{\rm D})$ and $(E_{\rm N}, E_{\rm D})$, respectively. A supervisor $S$ is called *fault-tolerant* if

$$S/L_{\rm FA} \subseteq E_{\rm A} \tag{7}$$

holds. □

Definition 5 states the *fault-tolerant control problem* and declares its *solution*.

*Definition 5.* A *fault-tolerant control problem* is a pair $((L_{\rm N}, L_{\rm D}), (E_{\rm N}, E_{\rm D}))$ with the fault-accommodating models $(L_{\rm N}, L_{\rm D})$ and $(E_{\rm N}, E_{\rm D})$. A solution to the fault-tolerant control problem is a non-empty formal language $K \subseteq \Sigma^*$ with

- $K \subseteq E_{\rm A}$
- $K$ is controllable w.r.t. $L_{\rm FA}$ and $\Sigma_{\rm uc}$
- $K$ is relatively closed w.r.t. $L_{\rm FA}$
- $\overline{K}$ is normal w.r.t. $\overline{L_{\rm FA}}$ and $\mathrm{p_o}$

□

Given a solution $K$ to the fault-tolerant control problem $((L_{\rm N}, L_{\rm D}), (E_{\rm N}, E_{\rm D}))$, the existence of a nonblocking, feasible, admissible and fault-tolerant supervisor $S$ such that $S/L_{\rm FA} = K$ is provided by [Ramadge and Wonham, 1989].

The implementation of fault-tolerant controllers as proposed in [Paoli et al., 2008], is simplified to the effect that there is no need for a diagnoser nor a supervisor switching mechanism. Furthermore, no additional tools for system analysis and control design need to be developed, since well-known algorithms (see libFAUDES [2006-2011]) for the solution of the supervisory control problem under partial observation can be reused.

The paper's *main result*, the construction of fault-tolerant supervisors for plants modelled by means of fault-accommodating models, is summarised in Procedure 1.

*Example 6.* Simple machine - supervisor design

Reconsider the simple machine from Ex. 3. The following safety requirements are imposed on the closed-loop behaviour.

*Procedure 1.* Computation of fault-tolerant supervisors

```
1: Set up the nominal model $L_{\rm N}$.
2: Determine possible faults and set up the plant's defective behaviour $L_{\rm D}$.
3: Compute the fault-accommodating model $L_{\rm FA}$ according to Eq. (4).
4: Set up the nominal acceptable behaviour $E_{\rm N}$.
5: Determine the defective plant's acceptable behaviour $E_{\rm D}$.
6: Compute the plant's acceptable behaviour $E_{\rm A}$ according to Eq. (6).
7: Compute the supremal controllable, normal and relatively closed sublanguage of $L_{\rm FA} \cap E_{\rm A}$.
```

□

(1) During nominal operation, the machine executes the production cycles A and B alternating and in this order.
(2) A fault occurs after the process A was started.
(3) After a fault has occurred, starting the process A is forbidden.

Figure 3 shows the acceptable nominal behaviour of the simple machine and the acceptable defective behaviour is shown in Fig. 4. Since the acceptable defective behaviour is proper and $E_{\rm N} \subseteq E_{\rm D}$ holds, the acceptable behaviour is given by the acceptable defective behaviour. Note that technically all events in $\Sigma - \{A, B\}$ are added by self-loops in every state of Fig. 3 and the event $a$ is added in Fig. 4, respectively.

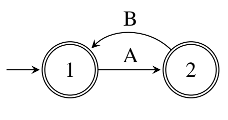


Fig. 3. Acceptable nominal behaviour $E_{\rm N}$

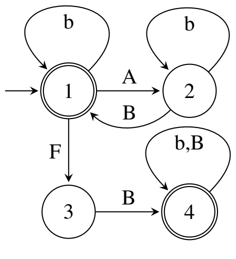


Fig. 4. Acceptable defective behaviour $E_{\rm A}$

The supremal controllable and normal sublanguage of the acceptable behaviour w.r.t. the plant's fault-accommodating behaviour and the controllable and observable events, respectively, can be computed using standard algorithms implemented in [libFAUDES, 2006-2011]. The result and its projection onto the observable events is shown in Figures 5 and 6.

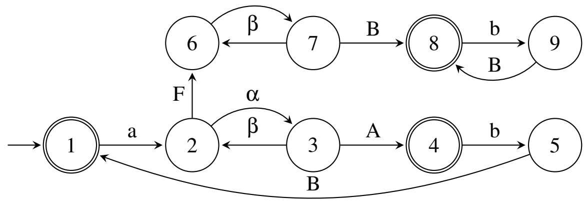


Fig. 5. Closed-loop system behaviour

Note that the closed-loop system is neither language diagnosable nor event diagnosable (Yoo and Garcia [2008]) w.r.t. to the nominal plant model. However, a fault-tolerant supervisor exists. □

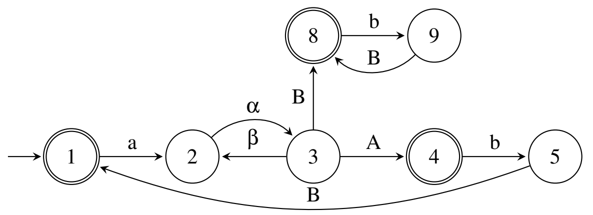


Fig. 6. Projection of closed-loop system to observable events

## 4. APPLICATION TO WORKPIECE FLOW CONTROL

This section provides the outline of an application example. Its implementation has been carried out and tested at a test facility of the Siemens AG.

A common situation in manufacturing automation is the collision-free control of a transportation system. Consider two conveyer belts, that merge into a single downstream segment as shown in Fig. 7.

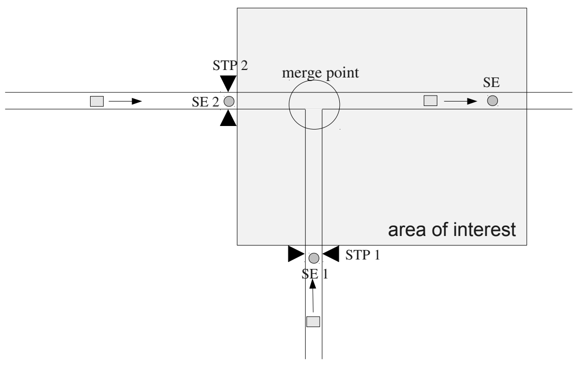


Fig. 7. The plant's physical layout

The conveyer belts are assumed to move at a fix speed and a change in direction is not possible. Besides the conveyer belts, the system possesses two stopper elements, STP1 and STP2, each one equipped with a sensor, SE1 and SE2, respectively. Stoppers are devices designed to subdivide a flow of workpieces, such that each workpiece can be treated independently. Possible collisions take place in the so-called *merge point*. Additionally, another sensor SE is located after the merge point.

Concerning fault-tolerant control, the plant is equipped with a redundant timer and the distance from STP2 to the merge point is assumed to be shorter than the distance from STP1 to the merge point.

System information can be extracted by evaluating the sensor states, and controller intervention is restricted to opening and closing the stoppers. After workpieces have reached the *area of interest*, controller actions have no impact on their movement, therefore, the controller's main task is the coordination of the workpiece entrance in the area of interest.

For simplicity only STP1 is subject to potential faults. A fault occurs, if STP1 becomes stuck in its open position. Its attributive sensor is taken to be functional in the presence of faults. Hence, STP1 is no longer available for active control intervention, but can still be used for sensor data aggregation.

### *4.1 Plant Model*

A. Stopper Model

The arrival of a workpiece at a stopper can be detected (*arrive*) by a sensor, attached to the stopper. Initially, the stopper is in a blocking state and an incoming workpiece is precluded from passing by. After a workpiece has been detected, the stopper is capable of either keeping the workpiece blocked, or opening up (*deblock*) and this way letting the workpiece pass. The *deblock*-event is associated with the stopper's complete opening and closing procedure. After the stopper has opened up, some time passes (*tau*) before the current workpiece passes by (*pass*) the stopper. A formal model for STP1 is shown in Fig. 8.

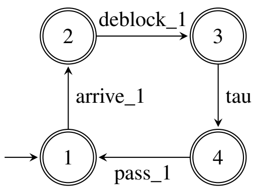


Fig. 8. Nominal stopper model

After the occurrence of a fault, the stopper's dynamics is no longer affected by the event *deblock*. A possible model for the stopper's defective behaviour is shown in Fig. 9. Note that Fig. 9 already shows the fault-accommodating behaviour computed according to Eq. (4).

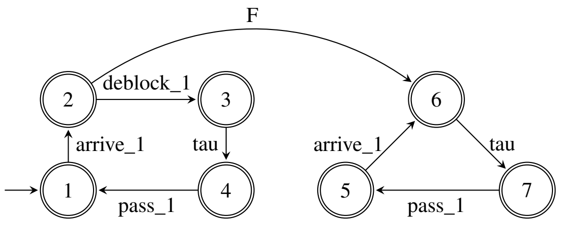


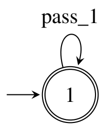
Fig. 9. Fault-accommodating stopper model

B. Sensor Model

The remaining sensor SE will solely report the detection of a workpiece (*se*). Fig. 10 shows the associated model.

C. Timer Model

The timer needs to elapse after the amount of time that is needed by a workpiece to travel from the broken stopper to the sensor SE. It is reset each time a workpiece passes STP1.



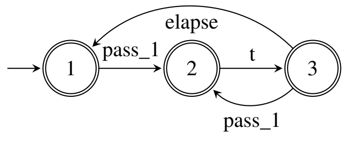


Fig. 10. Sensor model

Fig. 11. Timer model

Table 4.1 summarises the introduced events, their physical meaning and the attributes associated with them.

### *4.2 Controller Design*

The specification in Fig. 12 requires that if a workpiece has passed a stopper, the entrance of another workpiece in the area

Tab. 2 Physical meaning of events
C: controllable, O: observable, F: forcible

| Event | Interpretation | Attributes |
|---|---|---|
| arrive | workpiece detected at stopper | C, O |
| deblock | stopper performs open-close cycle | C, O |
| pass | workpiece passed the stopper | O |
| tau | passing of time | C, O, F |
| t | start timer | O |
| e | timer elapses | O |
| se | workpiece detected at sensor SE | O |
| F | fault | |

of interest is prevented until the sensor SE has detected the arrival of the workpiece. Note that technically all events in $\Sigma - \{pass\_1, pass\_2, se\}$ are accounted for by self-loops in every state but omitted for clarity in the graphical representation.

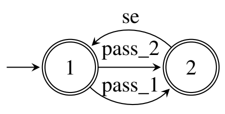


Fig. 12. Nominal safety specification

Once stopper STP1 is defective, the demanded safety needs might not be satisfied any more and the specification needs to be altered according to the fault's impact. Since STP2 is closer to the merge point than STP1, collisions can still be avoided by blocking all workpieces at STP2 at least for the period of time a workpiece needs to travel from STP1 to the merge point. This period of time is measured by the timer shown in Fig. 11. Since the timer serves as an indicator for a free area of intereset, the timer needs to be reset each time a workpiece passes the broken stopper. The automaton representation of the specification for the defective plant is shown in Fig. 13. Again all events in $\Sigma - \{pass\_1, pass\_2, se, e, F\}$ are accounted for by self-loops in every state, but omitted for clarity.

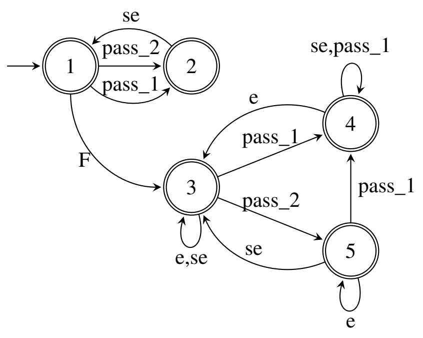


Fig. 13. A specification for the defective plant

The supremal normal and controllable sublanguage of the acceptable behaviour w.r.t. the plant's fault-accommodating behaviour and the controllable and observable events is non-empty and leads to a feasible controller implementation without the realisation of a diagnoser. Further, this implementation is guaranteed to respect the given design requirements under partial observation as well as the controller's restrictions w.r.t. the events' controllability properties. Computing this language using algorithms implemented in the software package [libFAUDES, 2006-2011] is straightforward. An automaton realisation for the fault-accommodating model exposes 63 states, five states for the specification and 81 states for the supervisor. This example will be available for download in form of a libfaudes Lua-Extension (see libFAUDES [2006-2011]) by the release-time of this paper.

## 5. CONCLUSION

The paper's main contribution is a modelling framework that accounts for the logical order of nominal events and faults. Its main result, the design of fault-tolerant supervisors for fault-accommodating models, is summarised in Procedure 1. These controllers can be implemented without an additional diagnosis tool or a switching mechanism. Since the fault-tolerant controller design problem is transformed to a standard supervisory control problem under partial observation, fault-tolerant supervisors for fault-accommodating models can be computed using well-known algorithms. Besides academic examples, a feasible application example from manufacturing automation is provided.

## REFERENCES

M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer, 2010.

S.E. Bourdon, M. Lawford, and W.M. Wonham. Robust nonblocking supervisory control of discrete-event systems. *IEEE Transactions of Automatic Control*, 50(12):2015–2021, 2005.

J.E.R. Cury and B.H. Krogh. Robustness of supervisors for discrete-event systems. *IEEE Transactions of Automatic Control*, 44(2):376–379, 1999.

libFAUDES. *Software library for discrete-event systems*, 2006-2011. Available at http://www.rt.eei.uni-erlangen.de/FGdes/faudes/index.html.

F. Lin. Robust and adaptive supervisory control of discrete event systems. *IEEE Transactions of Automatic Control*, 38 (12):1848–1852, 1993.

A. Paoli, M. Sartini, and S. Lafortune. A fault tolerant architecture for supervisory control of discrete event systems. In *Proceedings of the 17th IFAC world congress. Seoul. Korea.*, pages 6542–6547, 2008.

S.-J. Park and J.-T. Lim. Fault-tolerant robust supervisor for discrete event systems with model uncertainty and its application to a workcell. *IEEE Transactions on Robotics and Automation*, 15(2):386–391, 1999.

P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25(1):206–230, 1987.

P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

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

S. Takai. Robust supervisory control of a class of timed discrete event systems under partial observation. *Systems and Control Letters*, 39(4):267 – 273, 2000.

Q. Wen, R. Kumar, J. Huang, and H. Liu. A framework for fault-tolerant control for discrete event systems. *IEEE Transactions on Automatic Control*, 53(8):1839–1849, 2008.

W.M. Wonham. Supervisory control of discrete event systems. Monograph, 2009.

T.-S. Yoo and H.E. Garcia. Diagnosis of behaviors of interest in partially-observed discrete-event systems. *Systems and Control Letters*, 57(12):1023 – 1029, 2008.