2015 American Control Conference
Palmer House Hilton
July 1-3, 2015. Chicago, IL, USA

Fault Tolerant Controllability

Simon Radel, Aos Mulahuwaish, and Ryan J. Leduc

Abstract— **In this paper we investigate the problem of fault tolerance in the framework of discrete-event system (DES). We introduce our setting, and then provide a set of fault tolerant definitions designed to capture different types of fault scenarios and to ensure that our system remains controllable in each scenario. We then present algorithms to verify these properties. Finally, a simple example is provided to illustrate the properties.**

Keywords: **Discrete-event systems; Fault-tolerant; Supervisory control.**

I. Introduction

Supervisory control theory, introduced by Ramadge and Wonham [1], [2], [3], provides a formal framework for analysing discrete-event systems (DES). In this theory, automata are used to model the system to be controlled and the specification for the desired system behaviour. The theory provides methods and algorithms to obtain a supervisor that ensures the system will produce the desired behaviour.

However, the above typically assumes that the system behavior does not contain faults that would cause the actual system to deviate from the theoretical model. An example is a sensor that detects the presence of an approaching train. If the supervisor relies on this sensor to determine when the train should be stopped in order to prevent a collision, it could fail to enforce its control law if the sensor failed. Our goal in this paper is to develop a way to add fault events to the systems plant model and to categorize some common fault scenarios. We will then develop some properties that will allow us to determine if a supervisor will still be controllable in these scenarios.

Currently in the DES literature, the most common approach when a fault is detected is to switch to a new supervisor to handle the system in its degraded mode. Such an approach focuses on fault recovery as opposed to fault tolerance. This requires the construction of a second supervisor, and requires that there be a means to detect the occurrence of the fault in order to initiate the switch. In the approach we present in this paper, we use a single supervisor that will behave correctly in the presence of the specified fault scenarios. This method does not rely on detecting the fault, but on fault tolerant supervisors. We will now discuss some relevant previous work.

Lin [4] discussed both robust and adaptive supervisory control in discrete-event systems, including necessary and sufficient conditions for the existence of a robust supervisor. Based on this condition, a robust supervisory control and observation approach for synthesizing a supervisory control was developed. The goal of robust supervision is to synthesize a supervisor that realizes a given desired behavior for all possible systems.

In Park et al. [5], they presented necessary and sufficient conditions for fault tolerant robust supervisory control of discrete-event systems that belong to a set of models. When these conditions are satisfied, fault tolerance can be achieved. In the paper, the results were applied to the design, modelling, and control of a workcell consisting of arc welding (GMAW) robots, a sensor, and a conveyor.

In Paoli et al. [6], the controller was updated based on the information provided by online diagnostics. The supervisor needs to detect the malfunctioning component in the system in order to achieve the desired specification. The authors proposed the idea of safe diagnosability as a step to achieve the fault tolerant control. Two new notations were introduced in this work (safe controllability) and (active fault tolerant system), to characterize the conditions that must be satisfied when solving the fault tolerant control problem using this approach.

Qin Wen et al. [7] introduce a framework for fault-tolerant supervisory control of discrete-event systems. In this framework, plants contain both normal behavior and behavior with faults, as well as a submodel that contains only the normal behavior. The goal of fault-tolerant supervisory control is to enforce a specification for the normal behavior of the plant and to enforce another specification for the overall plant behavior. This includes ensuring that the plant recovers from any fault within a bounded delay so that after the recovery, the system state is equivalent to a state in the normal plant behavior. They formulate this notion of fault-tolerant supervisory control and provide a necessary and sufficient condition for the existence of such a supervisor. The condition involves notions of controllability, observability and relative-closure together with the notion of stability.

This paper is organized as follows. Section II discusses DES preliminaries. Section III introduces fault events, our fault tolerant controllability definitions and the fault scenarios to which they apply. Section IV presents algorithms to verify these properties. Section V provides a simple example. Finally, Section VI provides our conclusions.

Simon Radel is with the Dept. Génie Mécanique, ENS DE CACHAN, 61 avenue du Président Wilson, 94235 Cachan cedex, France sradel@ens-cachan.fr.gov

Aos Mulahuwaish and Ryan Leduc are with the Dept. of Computing and Software, McMaster University, Hamilton, ON, Canada, L8S 4K1 mulahuaa@mcmaster.ca and leduc@mcmaster.ca

II. PRELIMINARIES

We now present a summary of the DES terminology that we use in this paper. For more details, please refer to [3].

Let Σ be a finite set of distinct symbols (*events*), and Σ^* be the set of all finite sequences of events including ϵ, the *empty string*. Let Σ^+ denote the set of all finite, *non-empty* sequences of events. We can then define Σ^*:= $\Sigma^+ \cup \{\epsilon\}$.

Let $L \subseteq \Sigma^*$ be a *language* over Σ. A string $t \in \Sigma^*$ is a prefix of $s \in \Sigma^*$ (written $t \leq s$) if $s = tu$, for some $u \in \Sigma^*$. The *prefix closure* of language L (denoted $\overline{L}$) is defined as $\overline{L} := \{t \in \Sigma^* \mid t \leq s \text{ for some } s \in L\}$. Let Pwr($\Sigma$) denote the set of all possible subsets of Σ. For language L, the eligibility operator, $\text{Elig}_L : \Sigma^* \rightarrow \text{Pwr}(\Sigma)$, is given by $\text{Elig}_L(s) := \{\sigma \in \Sigma \,|s\sigma \in L \text{ for some } s \in \Sigma^*\}$.

A DES automaton is represented as a 5-tuple $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ where Y is the state set, Σ is the event set, the partial function $\delta : Y \times \Sigma \rightarrow Y$ is the transition function, y_o is the initial state, and Y_m is the set of marker states. The function δ is extended to $\delta : Y \times \Sigma^* \rightarrow Y$ in the natural way. The notation $\delta(y, s)!$ means that δ is defined for $s \in \Sigma^*$ at state y. For DES $\mathbf{G}$, the language generated is denoted by $L(\mathbf{G})$, and is defined to be $L(\mathbf{G}) := \{s \in \Sigma^* |\ \delta(y_o, s)!\}$. The marked behavior of $\mathbf{G}$ is defined as $L_m(\mathbf{G}) := \{\ s \in L(\mathbf{G})|\ \delta(y_o, s) \in Y_m\}$. The reachable state subset of DES $\mathbf{G}$, denoted Y_r, is $Y_r := \{y \in Y|\ (\exists s \in \Sigma^*)\, \delta(y_o, s) = y\}$. A DES $\mathbf{G}$ is reachable if $Y_r = Y$. We will always assume $\mathbf{G}$ is reachable.

Let $\Sigma = \Sigma_1 \cup \Sigma_2$, $L_1 \subseteq \Sigma_1^*$, and $L_2 \subseteq \Sigma_2^*$. For $i = 1, 2$, $s \in \Sigma^*$, and $\sigma \in \Sigma$, we define the *natural projection* $P_i : \Sigma^* \rightarrow \Sigma_i^*$ according to:

$$P_i(\epsilon) = \epsilon, \quad P_i(\sigma) = \begin{cases} \epsilon \text{ if } \sigma \notin \Sigma_i \\ \sigma \text{ if } \sigma \in \Sigma_i \end{cases}$$
$$P_i(s\sigma) = P_i(s)P_i(\sigma)$$

The map $P_i^{-1} : Pwr(\Sigma_i^*) \rightarrow Pwr(\Sigma^*)$ is the inverse image of P_i such that for $L \subseteq \Sigma_i^*$, $P_i^{-1}(L) := \{s \in \Sigma^* | P_i(s) \in L\}$.

Definition 1: For $\mathbf{G_i} = (Q_i, \Sigma_i, \delta_i, q_{o,i}, Q_{m,i})$ $(i = 1, 2)$, we define the *synchronous product* $\mathbf{G = G_1 || G_2}$ of the two DES as:

$$\mathbf{G} := (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{o,1}, q_{o,2}), Q_{m,1} \times Q_{m,2}),$$

where $\delta((q_1, q_2), \sigma)$ is only defined and equals:

(q_1', q_2') if $\sigma \in (\Sigma_1 \cap \Sigma_2), \delta_1(q_1, \sigma) = q_1', \delta_2(q_2, \sigma) = q_2'$ or
(q_1', q_2) if $\sigma \in \Sigma_1 - \Sigma_2, \delta_1(q_1, \sigma) = q_1'$or
(q_1, q_2') if $\sigma \in \Sigma_2 - \Sigma_1, \delta_2(q_2, \sigma) = q_2'$.

We note that if $\Sigma_1 = \Sigma_2$, we get $L(\mathbf{G}) = L(\mathbf{G_1}) \cap L(\mathbf{G_2})$.

For DES, the two main properties we want to check are *nonblocking* and *controllability*. A DES $\mathbf{G}$ is said to be *nonblocking* if $\overline{L_m(\mathbf{G})} = L(\mathbf{G})$. For controllability, we assume the standard event partition $\Sigma = \Sigma_u \dot{\cup} \Sigma_c$, splitting our alphabet into *uncontrollable* and *controllable events*.

Definition 2: A supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ is controllable for plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ if:

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u) s\sigma \in L(\mathbf{G}) \Rightarrow s\sigma \in L(\mathbf{S})$$

We now provide some language definitions that will be useful for this paper. We start with the language L^k. This is the set of strings constructed from any k strings in L.

Definition 3: Let $L \subseteq \Sigma^*$ and $k \in \{1, 2, \ldots\}$. We define the language L^k to be:

$$L^k := \{s \in \Sigma^* | s = s_1 s_2 ... s_k \text{ for some } s_1, s_2, ..., s_k \in L\}$$

We next define the notation for the language constructed from all possible ways to concatenate a string from the first language, followed by an event from the event set, and a string from the second language.

Definition 4: Let L_1, $L_2 \subseteq \Sigma^*$ and $\Sigma' \subseteq \Sigma$. We define the language $L_1.\Sigma'.L_2$ to be:
$L_1.\Sigma'.L_2 := \{s \in \Sigma^* | s = s_1 \sigma s_2 \text{ for some } s_1 \in L_1, s_2 \in L_2, \sigma \in \Sigma'\}$

III. FAULT TOLERANT SUPERVISORS

In this section, we will introduce our concept of fault events and then categorize some common fault scenarios. We will then develop some properties that will allow us to determine if a supervisor will still be controllable in these scenarios. We will assume that all DES are deterministic, and that we are given plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ and supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$.

A. Fault Events

In this paper, our approach will be to add a set of uncontrollable events to our plant model to represent the possible faults in the system. For example, if we had a sensor to detect when a train passes, its plant model might originally contain an event such as *trn_sen0* indicating a train is present. We could add a new uncontrollable event, *trnf_sen0*, that will occur instead if the sensor fails to detect the train. This will allow us to model how the system will behave after the occurrence of the fault. Our goal will be to design supervisors that will still behave correctly even if a fault event occurs, even though they can't detect the fault event directly.

We start by defining a group of $m \geq 0$ mutually exclusive sets of *fault events*. The idea here is to group related faults into sets such that faults of a given set represent a common fault condition, while faults of a different set represent a different fault condition. For example, two sensors in a row that could each be used to detect the train in time for a given track segment might be in the same fault set, but a sensor in a different part of the track would be in a different set.

$$\Sigma_{F_i} \subseteq \Sigma_u, i = 1, .., m$$

Definition 5: We define the set of *standard fault events*, Σ_F, as:

$$\Sigma_F := \dot{\bigcup_{i=1}^{m}} \Sigma_{F_i}$$

We note that for $m = 0$, $\Sigma_F = \emptyset$.

The standard fault events are the faults that will be used to define the various fault scenarios that our supervisors will need to be able to handle. However, there are two additional types of faults that we need to define to handle two special cases. The first type is called *unrestricted fault events*, denoted $\Sigma_{\Omega F} \subseteq \Sigma_u$. These are faults that a supervisor can always handle and thus are allowed to occur unrestricted.

The second type is called *excluded fault events*, denoted $\Sigma_{\Delta F} \subseteq \Sigma_u$. These are faults that can not be handled at all and thus are essentially ignored in our scenarios. The idea is that this would allow us to still design a fault tolerant supervisor for the remaining faults. Typically, most systems would have neither excluded or unrestricted faults, but we will include them in our definitions for the systems that do.

For each fault set, Σ_{F_i} $(i = 0, .., m)$, we also need to define a matching set of *reset events*, denoted $\Sigma_{T_i} \subseteq \Sigma$. These events will be explained in Section III-C.

B. Fault Tolerant Consistency

We now present a consistency requirement that our systems must satisfy.

Definition 6: A system, with plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, and fault sets Σ_{F_i} $(i = 0, .., m)$, $\Sigma_{\Delta F}$, and $\Sigma_{\Omega F}$, is *fault tolerant (FT) consistent* if:

1) $\Sigma_{\Delta F} \cup \Sigma_{\Omega F} \cup \Sigma_F \subseteq \Sigma_u$
2) $\Sigma_{\Delta F}, \Sigma_{\Omega F}, \Sigma_{F_i} (i = 0, .., m)$, are pair-wise disjoint.
3) $(\forall i \in 1, .., m)\Sigma_{F_i} \neq \emptyset$
4) $(\forall i \in 1, .., m)\Sigma_{F_i} \cap \Sigma_{T_i} = \emptyset$
5) Supervisor $\mathbf{S}$ is deterministic.
6) $(\forall x \in X)(\forall \sigma \in (\Sigma_{\Omega F} \cup \Sigma_{\Delta F} \cup \Sigma_F))\xi(x, \sigma) = x$

Point (1) says that fault events are uncontrollable since allowing a supervisor to disable fault events would be unrealistic. **Point (2)** requires that the indicated sets of faults be disjoint since they must each be handled differently. **Point (3)** says that fault sets Σ_{F_i} are non-empty. **Point (4)** says a fault set must be disjoint from its corresponding set of reset events.

Points (5) and **(6)** say that $\mathbf{S}$ is deterministic (single initial state and at most a single transition leaving a given state for a given event) and that at every state in $\mathbf{S}$, there is a selfloop for each fault event in the system. This means a supervisor cannot change state (and thus change enablement information) based on a fault event. This is a key concept as it effectively makes fault events unobservable to supervisors. If $\mathbf{S}$ is defined over a subset $\Sigma' \subset \Sigma$ instead, we could equivalently require that Σ' contain no fault events.

C. Fault Scenarios

In this paper, we will consider four fault scenarios. The first is the *default fault scenario* where the supervisor must be able to handle any non-excluded fault event that occurs. The second scenario is the $N \geq 0$ *fault scenario* where the supervisor is only required to handle at most N, non-excluded fault events and all unrestricted fault events.

The next scenario is the *non-repeatable* $N \geq 0$ *fault scenario* where the supervisor is only required to handle at most N, non-excluded fault events and all unrestricted fault events, but no more than one fault event from any given Σ_{F_i} $(i = 0, .., m)$ fault set. This definition allows the designer to group faults together in fault sets such that a fault occurring from one set does not affect a supervisors ability to handle a fault from a different set. Particularly for a situation where a supervisor could handle only one fault per fault set, this would allow m faults to occur instead of only one using the previous scenario.

The last scenario we consider is the *resettable fault scenario.* This is designed to capture the situation where at most one fault event from each Σ_{F_i} $(i = 0, .., m)$ fault set can be handled by the supervisor during each pass through a part of the system, but this ability resets for the next pass. For this to work, we need to be able to detect when the current pass has completed and it is safe for another fault event from the same fault set to occur. We use the fault set's corresponding set of reset events to achieve this. The idea is that once a reset event has occurred, the current pass can be considered over and it is safe for another fault event to occur.

D. Fault Tolerant Controllability

The first fault tolerant property that we introduce is designed to handle the default fault scenario. First, we need to define the *language of excluded faults.* This is the set of all strings that include at least one fault from $\Sigma_{\Delta F}$.

Definition 7: We define the *language of excluded faults* as:

$$L_{\Delta F} := \Sigma^*.\Sigma_{\Delta F}.\Sigma^*$$

Definition 8: A system, with plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, and fault sets Σ_{F_i} $(i = 0, .., m)$ and $\Sigma_{\Delta F}$, is *fault tolerant (FT) controllable* if it is FT consistent and:

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u) \\ (s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F}) \Rightarrow s\sigma \in L(\mathbf{S})$$

The above definition is essentially the standard controllability definition, but ignores strings that include excluded fault events. As the language $L(\mathbf{S}) \cap L(\mathbf{G})$ is prefix closed, prefixes of these strings that do not contain excluded faults must be checked. This definition is equivalent to blocking all excluded fault events from occurring in the system behavior and then checking the standard controllability definition. This is the most powerful of the fault tolerant definitions as the supervisor must be able to handle a potentially unlimited number of faults that can occur in any order and at any time.

Typically, the set of excluded faults for a given system is empty. When a system is FT controllable and $\Sigma_{\Delta F} \neq$

$\emptyset$, we say that it is *FT controllable with excluded faults* to emphasize that it is less fault tolerant than if it passed the definition with $\Sigma_{\Delta F} = \emptyset$. We will use a similar expression with the other fault tolerant definitions.

E. N-Fault Tolerant Controllability

The next fault tolerant property that we introduce is designed to handle the $N \geq 0$ fault scenario. First, we need to define the *language of N-fault events.* This is the set of all strings that include at most N faults from fault sets Σ_{F_i} $(i = 0, .., m)$, including those that contain no such faults.

Definition 9: We define the *language of N-fault events* as:

$$L_{NF} := (\Sigma - \Sigma_F)^* \cup \bigcup_{k=1}^{N} ((\Sigma - \Sigma_F)^*.\Sigma_F.(\Sigma - \Sigma_F)^*)^k$$

Definition 10: A system, with plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, and fault sets Σ_{F_i} $(i = 0, .., m)$ and $\Sigma_{\Delta F}$, is *N-fault tolerant (N-FT) controllable* if it is FT consistent and:

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)$$
$$(s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F}) \wedge (s \in L_{NF}) \Rightarrow s\sigma \in L(\mathbf{S})$$

The above definition is essentially the standard controllability definition, but ignores strings that include excluded fault events or more than N faults from Σ_F. This definition is essentially weaker than the previous one since if we take $N = \infty$ we get the FT controllability definition back. If we set $N = 0$, we get the controllability definition with all fault events from Σ_F excluded as well.

Typically, the set of unrestricted faults for a given system is empty. When a system is N-FT controllable and $\Sigma_{\Omega F} \neq \emptyset$, we say that it is *N-FT controllable with unrestricted faults* to emphasize that it is more fault tolerant than if it passed the definition with $\Sigma_{\Omega F} = \emptyset$. We will use a similar expression with the other fault tolerant definitions.

F. Non-repeatable N-Fault Tolerant Controllability

The next fault tolerant property that we introduce is designed to handle the non-repeatable $N \geq 0$ fault scenario. First, we need to define the *language of non-repeatable fault events.* This is the set of all strings that include two or more faults from a single fault set Σ_{F_i} $(i = 0, .., m)$.

Definition 11: We define the *language of non-repeatable fault events* as:

$$L_{NRF} := \bigcup_{i=1}^{m} (\Sigma^*.\Sigma_{F_i}.\Sigma^*.\Sigma_{F_i}.\Sigma^*)$$

Definition 12: A system, with plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, and fault sets Σ_{F_i} $(i = 0, .., m)$ and $\Sigma_{\Delta F}$, is *non-repeatable N-fault tolerant (NR-FT) controllable*, if it is FT consistent and:

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)(s\sigma \in L(\mathbf{G})) \wedge$$
$$(s \notin (L_{\Delta F} \cup L_{NRF})) \wedge (s \in L_{NF}) \Rightarrow s\sigma \in L(\mathbf{S})$$

The above definition is essentially the standard controllability definition, but ignores strings that include excluded fault events, more than N faults from Σ_F, or strings that include two or more faults from a single fault set.

G. Resettable Fault Tolerant Controllability

The next fault tolerant property that we introduce is designed to handle the resettable fault scenario. First, we need to define the *language of resettable fault events.* This is the set of all strings where two faults from the same fault set Σ_{F_i} occur in a row without an event from the corresponding set of reset events in between.

Definition 13: We define the *language of resettable fault events* as:

$$L_{TF} := \bigcup_{i=1}^{m} (\Sigma^*.\Sigma_{F_i}.(\Sigma - \Sigma_{T_i})^*.\Sigma_{F_i}.\Sigma^*)$$

Definition 14: A system, with plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, fault and reset sets $\Sigma_{F_i}, \Sigma_{T_i}$ $(i = 0, .., m)$ and fault set $\Sigma_{\Delta F}$, is *resettable fault tolerant (T-FT) controllable* if it is FT consistent and:

$$(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}))(\forall \sigma \in \Sigma_u)$$
$$(s\sigma \in L(\mathbf{G})) \wedge (s \notin L_{\Delta F} \cup L_{TF}) \Rightarrow s\sigma \in L(\mathbf{S})$$

The above definition is essentially the standard controllability definition, but ignores strings that include excluded fault events and strings where we get two fault events from the same fault set in a row without an event from the corresponding set of reset events in between.

IV. Algorithms

In this section, we will present algorithms to verify the four fault tolerant controllability properties that we defined in Section III. We will not present an algorithm for the FT consistency property as its individual points can easily be checked by adapting various standard algorithms. We assume that our system consists of a plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, supervisor $\mathbf{S} = (X, \Sigma, \eta, x_o, X_m)$, and fault and reset sets $\Sigma_{F_i}, \Sigma_{T_i}$ $(i = 0, .., m)$, $\Sigma_{\Delta F}$, and $\Sigma_{\Omega F}$.

Our approach will be to construct plant components to synchronize with our plant $\mathbf{G}$ such that the new DES will restrict the occurrence of faults to match the given fault tolerant controllability definition. We can then synchronize the plant components together and then use a standard controllability algorithm to check the property. This approach allows us to automatically take advantage of existing scalability methods such as incremental [8], and binary decision diagram-based (BDD) algorithms [9], [10], [11], [12], [13], [14].

As the controllability and synchronous product have already been studied in the literature [15], we will assume that they are given to us. We will use the standard $||$ to indicate the synchronous product operation, and *vCont*(**Plant**,**Sup**) to indicate controllability verification. Function *vCont* returns *true* or *false* to indicate whether the verification passed or failed, and the result will be stored in the Boolean variable *pass*.

A. Verify Fault Tolerant Controllability

Algorithm 1 shows how to verify fault tolerant controllability for $\mathbf{G}$ and $\mathbf{S}$. Line 1 constructs the *excluded fault plant*, $\mathbf{G_{\Delta F}}$, using **Algorithm 2**. Line 2 constructs the new plant $\mathbf{G}'$. Line 3 checks that supervisor $\mathbf{S}$ is controllable for plant $\mathbf{G}'$. As $\mathbf{G_{\Delta F}}$ is defined over event set $\Sigma_{\Delta F}$ and contains only a marked initial state and no transitions, synchronizing it with $\mathbf{G}$ creates the original behavior with all excluded fault events removed. Checking that $\mathbf{S}$ is controllable for the resulting behavior will have the effect of verifying fault tolerant controllability.

Algorithm 1 Verify fault tolerant controllability

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$;
2: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}$;
3: pass $\leftarrow$ vCont$(\mathbf{G}', \mathbf{S})$;
4: **return** pass;

Algorithm 2 constructs $\mathbf{G_{\Delta F}}$ for fault set $\Sigma_{\Delta F}$. The algorithm constructs a new DES with event set $\Sigma_{\Delta F}$, but no transitions. It also contains only its initial state, which is marked. This will have the effect of removing any $\Sigma_{\Delta F}$ transitions from any DES it is synchronized with.

Please note that all of the constructed DES in these algorithms always have all their states marked since their goal is to modify the closed behavior by restricting the occurrence of fault events as needed, not to modify the marked behavior of the system directly. Also, when we define our transition functions such as δ_1, we will define them as a subset of $Y \times \Sigma \times Y$ for convenience.

Algorithm 2 construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$

1: $Y_1 \leftarrow \{y_0\}$;
2: $Y_{m,1} \leftarrow Y_1$;
3: $\delta_1 \leftarrow \emptyset$;
4: **return** $(Y_1, \Sigma_{\Delta F}, \delta_1, y_o, Y_{m,1})$;

Figure 1 shows an example $\mathbf{G_{\Delta F}}$. In the DES diagrams, circles represent unmarked states, while filled circles represent marked states. Two concentric, unfilled circles represent the initial state. If the initial state is also marked, the inner circle is filled. Uncontrollable event labels are preceded by an "!". Note that if a transition is labeled by an event set such as in Figure 2, this is a shorthand for a transition for each event in the event set.

B. Verify N-Fault Tolerant Controllability

Algorithm 3 shows how to verify N-fault tolerant controllability for $\mathbf{G}$, and $\mathbf{S}$. Line 1 constructs the excluded fault plant, $\mathbf{G_{\Delta F}}$. Line 2 constructs the *N-fault plant*, $\mathbf{G_{NF}}$, using **Algorithm 4**. Line 3 constructs the new plant $\mathbf{G}'$. Line 4 checks that supervisor $\mathbf{S}$ is controllable for plant $\mathbf{G}'$. As $\mathbf{G_{\Delta F}}$ removes any excluded fault transitions and $\mathbf{G_{NF}}$ prevents strings from containing more than N fault events,

Fig. 1. Excluded Fault Plant $\mathbf{G_{\Delta F}}$

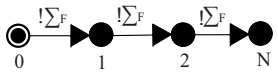

Fig. 2. N-Fault Plant $\mathbf{G_{NF}}$, N = 3

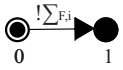

Fig. 3. Non-Repeatable N-Fault Plant $\mathbf{G_{F,i}}$

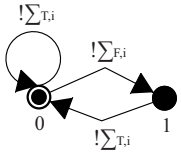

Fig. 4. Resettable Fault Plant $\mathbf{G_{TF,i}}$

checking that $\mathbf{S}$ is controllable for the resulting behavior will have the effect of verifying N-fault tolerant controllability.

Algorithm 3 Verify N-fault tolerant controllability

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$;
2: $\mathbf{G_{NF}} \leftarrow$ construct-$\mathbf{G_{NF}}(N, \Sigma_F)$;
3: $\mathbf{G}' \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}$;
4: pass $\leftarrow$ vCont$(\mathbf{G}', \mathbf{S})$;
5: **return** pass;

Algorithm 4 constructs $\mathbf{G_{NF}}$ for max N faults and standard fault set Σ_F. The algorithm constructs a new DES with event set Σ_F and $N+1$ states, each state marked. It then creates a transition for each fault event in Σ_F from state y_i to state y_{i+1}. As there are no transitions at state y_N, synchronizing with this DES will allow at most N faults to occur, and then remove any additional standard fault transitions. Figure 2 shows an example $\mathbf{G_{NF}}$ for $N = 3$.

Algorithm 4 construct-$\mathbf{G_{NF}}(N, \Sigma_F)$

1: $Y_1 \leftarrow \{y_0, y_1, ..., y_N\}$;
2: $Y_{m,1} \leftarrow Y_1$;
3: $\delta_1 \leftarrow \emptyset$;
4: **for** $i = 0, ..., N-1$;
5: **for** $\sigma \in \Sigma_F$;
6: $\delta_1 \leftarrow \delta_1 \cup \{(y_i, \sigma, y_{i+1})\}$;
7: **end for**;
8: **end for**;
9: **return** $(Y_1, \Sigma_F, \delta_1, y_o, Y_{m,1})$;

C. Verify Non-repeatable N-Fault Tolerant Controllability

Algorithm 5 shows how to verify non-repeatable N-fault tolerant controllability for $\mathbf{G}$ and $\mathbf{S}$. Line 1 constructs the excluded fault plant, $\mathbf{G_{\Delta F}}$. Line 2 constructs the N-fault plant, $\mathbf{G_{NF}}$. For $i \in \{1, \ldots, m\}$, Line 4 constructs the *non-repeatable N-fault plant*, $\mathbf{G_{F,i}}$, using **Algorithm 6**. Line 6 constructs the new plant $\mathbf{G}'$. Line 7 checks that supervisor $\mathbf{S}$ is controllable for plant $\mathbf{G}'$. As $\mathbf{G_{\Delta F}}$ removes any excluded fault transitions, $\mathbf{G_{NF}}$ prevents strings from containing more than N fault events, and each $\mathbf{G_{F,i}}$ allows at most one fault from their fault set to occur, checking that $\mathbf{S}$ is controllable for the resulting behavior will have the effect of verifying

non-repeatable N-fault tolerant controllability. We note that if $m \leq N$, we can safely skip Line 2 as Lines 3-5 will ensure at most m faults can occur.

Algorithm 5 Verify non-repeatable N-fault tolerant controllability

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$;
2: $\mathbf{G_{NF}} \leftarrow$ construct-$\mathbf{G_{NF}}(N, \Sigma_F)$;
3: **for** $i = 1, \ldots, m$;
4: $\quad \mathbf{G_{F,i}} \leftarrow$ construct-$\mathbf{G_{F,i}}(\Sigma_{F_i}, i)$;
5: **end for**;
6: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{NF}}||\mathbf{G_{F,1}}||.....||\mathbf{G_{F,m}}$;
7: pass $\leftarrow$ vCont$(\mathbf{G'}, \mathbf{S})$;
8: **return** pass;

Algorithm 6 constructs $\mathbf{G_{F,i}}$ for $i \in \{1, \ldots, m\}$, and fault set Σ_{F_i}. The algorithm constructs a new DES with event set Σ_{F_i} and two states, each state marked. It then creates a transition for each fault event in Σ_{F_i} from the initial state to state y_1. As there are no transitions at state y_1, synchronizing with this DES will allow at most 1 fault event from the fault set to occur, and then remove any additional fault transitions from the fault set. Figure 3 shows an example $\mathbf{G_{F,i}}$.

Algorithm 6 construct-$\mathbf{G_{F,i}}(\Sigma_{F_i}, i)$

1: $Y_i \leftarrow \{y_0, y_1\}$;
2: $Y_{m,i} \leftarrow Y_i$;
3: $\delta_i \leftarrow \emptyset$;
4: **for** $\sigma \in \Sigma_{F_i}$;
5: $\quad \delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$;
6: **end for**;
7: **return** $(Y_i, \Sigma_{F_i}, \delta_i, y_o, Y_{m,i})$;

D. Verify Resettable Fault Tolerant Controllability

Algorithm 7 shows how to verify resettable fault tolerant controllability for $\mathbf{G}$ and $\mathbf{S}$. Line 1 constructs the excluded fault plant. For $i \in \{1, \ldots, m\}$, Line 3 constructs the *resettable fault plant* $\mathbf{G_{TF,i}}$, using **Algorithm 8**. Line 5 constructs the new plant $\mathbf{G'}$. Line 6 checks that supervisor $\mathbf{S}$ is controllable for plant $\mathbf{G'}$. As $\mathbf{G_{\Delta F}}$ removes any excluded fault transitions, and each $\mathbf{G_{TF,i}}$ only allows strings where fault events from Σ_{F_i} are always separated by at least one event from the corresponding set of reset events, Σ_{T_i}, checking that $\mathbf{S}$ is controllable for the resulting behavior will have the effect of verifying resettable fault tolerant controllability.

Algorithm 7 Verify resettable fault tolerant controllability

1: $\mathbf{G_{\Delta F}} \leftarrow$ construct-$\mathbf{G_{\Delta F}}(\Sigma_{\Delta F})$;
2: **for** i = 1,...,m;
3: $\quad \mathbf{G_{TF,i}} \leftarrow$ construct-$\mathbf{G_{TF,i}}(\Sigma_{F_i}, \Sigma_{T_i}, i)$;
4: **end for**;
5: $\mathbf{G'} \leftarrow \mathbf{G}||\mathbf{G_{\Delta F}}||\mathbf{G_{TF,1}}||.....||\mathbf{G_{TF,m}}$;
6: pass $\leftarrow$ vCont$(\mathbf{G'}, \mathbf{S})$;
7: **return** pass;

Algorithm 8 constructs $\mathbf{G_{TF,i}}$ for $i \in \{1, \ldots, m\}$, fault set Σ_{F_i}, and the set of reset events, Σ_{T_i}. The algorithm constructs a new DES with event set $\Sigma_{F_i} \cup \Sigma_{T_i}$ and two states, each state marked. It then creates a transition for each fault event in Σ_{F_i} from the initial state to state y_1. Next, it creates a transition for each reset event in Σ_{T_i} from state y_1 to the initial state, as well as a selfloop at the initial state. Figure 4 shows an example $\mathbf{G_{TF,i}}$. Essentially, reset events can occur unrestricted, but once a fault event occurs from Σ_{F_i}, a second event from the set is blocked until a reset event from Σ_{T_i} occurs. Synchronizing with this DES will have the effect of restricting the plant's fault behavior to that which the supervisor is required to handle.

Algorithm 8 construct-$\mathbf{G_{TF,i}}(\Sigma_{F_i}, \Sigma_{T_i}, i)$

1: $Y_i \leftarrow \{y_0, y_1\}$;
2: $Y_{m,i} \leftarrow Y_i$;
3: $\delta_i \leftarrow \emptyset$;
4: **for** $\sigma \in \Sigma_{F_i}$;
5: $\quad \delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_1)\}$;
6: **end for**;
7: **for** $\sigma \in \Sigma_{T_i}$;
8: $\quad \delta_i \leftarrow \delta_i \cup \{(y_0, \sigma, y_0), (y_1, \sigma, y_0)\}$;
9: **end for**;
10: **return** $(Y_i, \Sigma_{F_i} \cup \Sigma_{T_i}, \delta_i, y_o, Y_{m,i})$;

V. Manufacturing Testbed Example

This example is based on the manufacturing testbed from Leduc [16]. The testbed was designed to simulate a manufacturing workcell, in particular problems of routing and collision.

In this paper, we will focus on only a single track loop, shown in Figure 5. The loop contains 8 sensors and two trains (*train1*, *train2*). Train1 starts between sensors 9 and 10, while train 2 starts between sensors 15 and 16. Both trains can only traverse the tracks in a clockwise direction.

A. Plant Models

The plant model for the testbed consists of the following basic elements: sensors, trains and the relationship between sensors and trains.

1) Sensor Models: The sensor models indicate when a given train is present and when no trains are present. Also, they state that only one train can activate a given sensor at

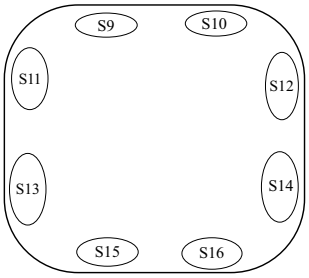

Fig. 5. Single Train Loop

a time. Figure 8 shows the original sensor model, for sensor $J \in \{9, \ldots, 16\}$.

To add faults to the model, we assumed that sensors 9 10, and 16 could have an intermittent fault; sometimes the sensor would detect the presence of a train, sometimes it would fail to do so. We modelled this by adding to all the plant models a new event *t1F_atJ*, $J \in \{9, 10, 16\}$, for each *t1_atJ* event. For each *t1_atJ* transition in a plant model, we added an identical *t1F_atJ* transition. The idea is we can now get the original detection event or the new fault one instead. We made similar changes for train2. Figure 9 shows the new sensor models with the added fault events.

For this example, $\Sigma_{\Delta F} = \Sigma_{\Omega F} = \emptyset$. We also set $m = 2$, $\Sigma_{F_1} = \{t1F\_at9, t1F\_at10\}$, $\Sigma_{F_2} = \{t1F\_at16\}$, $\Sigma_{T_1} = \{t1\_at11\}$, and $\Sigma_{T_2} = \{t1\_at14\}$.

2) Sensor Interdependencies: This series of models show the sensor's interdependencies with respect to a given train. With respect to the starting position of a particular train (represented by the initial state), sensors can only be reached in a particular order, dictated by their physical location on the track. This is shown in Figures 6 and 7. Both DES already show the added fault events.

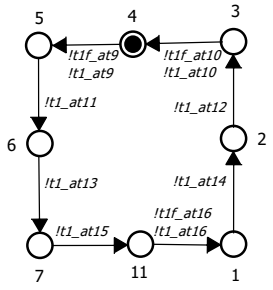

Fig. 6. Sensor Interdependencies For Train 1

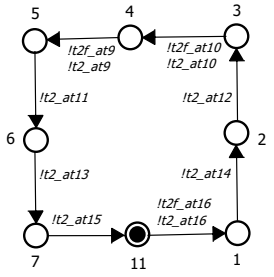

Fig. 7. Sensor Interdependencies For Train 2

3) Train Models: The train models are shown in Figure 10 for trainK ($K = 1, 2$). TrainK can only move when its enablement event, *en_trainK*, occurs, and then it can move at most a single unit of distance (event *umv_trainK*), before another *en_trainK* must occur. This allows a supervisor to precisely control the movement of the train by enabling and disabling event *en_trainK* as needed.

4) Relationship Between Sensors and Trains Models: Figure 11 shows the relationship between trainK's ($K = 1, 2$) movement, and a sensor detecting the train. It captures the idea that a train can reach at most one sensor during a unit movement, and no sensors if it is disabled. Figure 12 shows the model with fault events added.

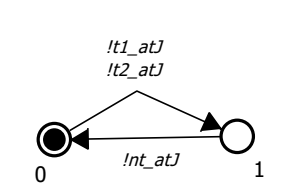

Fig. 8. Original Sensor Model

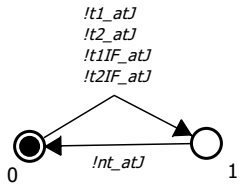

Fig. 9. Sensors 9, 10, and 16 with Faults

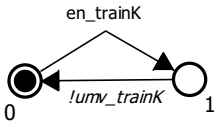

Fig. 10. Train Models

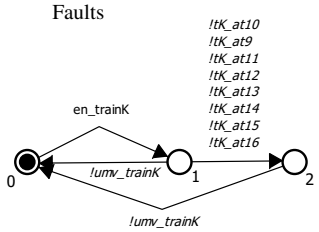

Fig. 11. Sensors and Trains

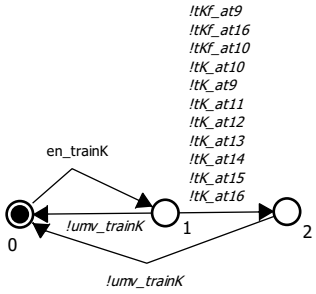

Fig. 12. Sensors and Trains with Faults

B. Modular Supervisor

After the plant models were developed, a single supervisor was designed to prevent collisions in the track section with sensors 11 and 13. The idea is to ensure that only one train uses this track section at a time. We will first introduce the original collision protection supervisor that was designed with the assumption of no faults, and then we will introduce a fault tolerant supervisor.

1) Collision Protection Supervisor: Figure 13 shows the collision protection supervisor (CPS) for the track section containing sensors 11 and 13. Once a train has reached sensor 11, the other train is stopped at sensor 10 until the first train reaches sensor 15, which indicates it has left the protected area. The stopped train is then allowed to continue.

It's easy to see that this supervisor will fail all four fault tolerant controllability definitions as it relies solely on sensor 10 to detect when a second train arrives. If sensor 10 fails, the train continues and could collide with the first train.

2) Collision Protection Fault Tolerant Supervisor: We next modified the supervisor in Figure 13 to make it more fault tolerant. The result is shown in Figure 14. We have added at state 1 a check for either sensor 9 or sensor 10.

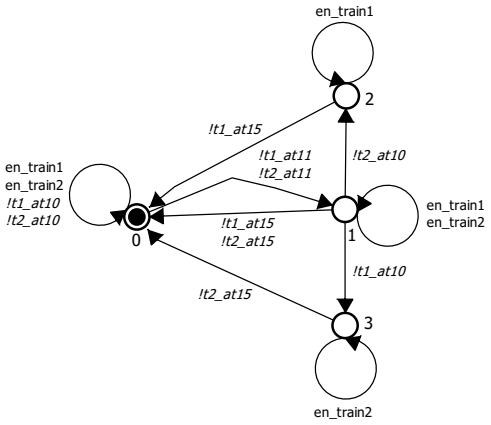

Fig. 13. CPS Supervisor

That way if sensor 10 fails but sensor 9 doesn't, we can still stop the train at sensor 9 and avoid the collision.

Using the DES research software tool, DESpot [17], we can verify that the supervisor is not fault tolerant controllable for the plant. The reason is that if both sensors 9 and 10 fail, the train will not stop and a collision could occur. However, the system can be show to be N-fault tolerant controllable for $N = 1$ (sensor 10 fails but not sensor 9), non-repeatable N-fault tolerant controllable for $N = 2$ (a fault from Σ_{F_2} doesn't affect anything), and resettable fault tolerant controllable (as long as both sensors 9 and 10 don't fail in a given pass, all is well).

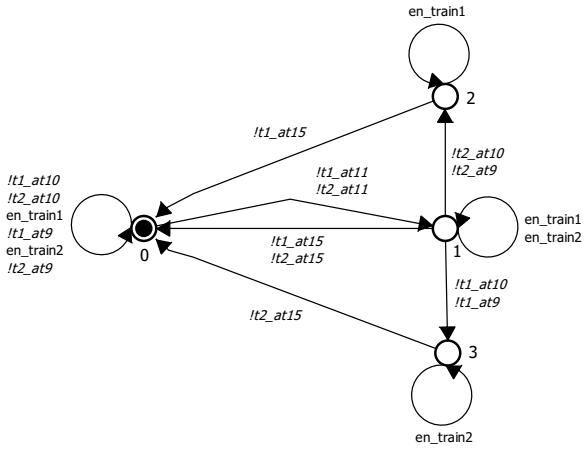

Fig. 14. CPS With Redundancy for Faults

VI. Conclusions

In this paper we investigate the problem of fault tolerance in the framework of discrete-event system (DES). We introduce a set of four fault tolerant controllability definitions designed to capture different types of fault scenarios and to ensure that our system remains controllable in each scenario. We then present a set of algorithms to verify the properties. As these algorithms involve adding new plant components and then checking standard controllability, they can instantly take advantage of existing controllability algorithms, software, and scalability approaches such as incremental verification and binary decision diagrams (BDD). We finished with a small example that illustrated how the theory can be applied.

References

[1] P. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete-event processes," *SIAM J. Control Optim*, vol. 25, no. 1, pp. 206–230, 1987.

[2] W. M. Wonham and P. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM J. Control Optim*, vol. 25, no. 3, pp. 637–659, May 1987.

[3] W. M. Wonham, *Supervisory Control of Discrete-Event Systems*, Department of Electrical and Computer Engineering, University of Toronto, July 2014, Monograph and TCT software can be downloaded at http://www.control.toronto.edu/DES/.

[4] F. Lin, "Robust and adaptive supervisory control of discrete event systems," *IEEE Trans. Automatic Control*, vol. 38, no. 12, pp. 1848–1852, Dec. 1993.

[5] 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*, vol. 15, no. 2, pp. 386–391, 1999.

[6] A. Paoli, M. Sartini, and S. Lafortune, "Active fault tolerant control of discrete event systems using online diagnostics," *Automatica*, vol. 47, no. 4, pp. 639–649, 2011.

[7] Q. Wen, R. Kumar, J. Huang, and H. Liu, "A framework for fault-tolerant control of discrete event systems," *IEEE Trans. on Automatic Control*, vol. 53, pp. 1839–1849, 2008.

[8] B. A. Brandin, R. Malik, and P. Malik, "Incremental verification and synthesis of discrete-event systems guided by counter-examples," *IEEE Trans. on Control Systems Technology*, vol. 12, no. 3, pp. 387–401, May 2004.

[9] A. E. Bryant, "Symbolic boolean manipulation with ordered binary-decision diagrams," *ACM Computing Surveys*, vol. 24, pp. 293–318, 1992.

[10] Z. Zhang, "Smart TCT: an efficient algorithm for supervisory control design." Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 2001.

[11] C. Ma, "Nonblocking supervisory control of state tree structures," Ph.D. dissertation, Department of Electrical and Computer Engineering, University of Toronto, 2004.

[12] A. Vahidi, B. Lennartson, and M. Fabian, "Efficient analysis of large discrete-event systems with binary decision diagrams," in *Proc. of the 44th IEEE Conf. Decision Contr. and and 2005 European Contr. Conf.*, Seville, Spain, 2005, pp. 2751–2756.

[13] R. Song, "Symbolic synthesis and verification of hierarchical interface-based supervisory control," Master's thesis, Dept. of Comput. and Softw., McMaster University, Hamilton, Ont, 2006.

[14] Y. Wang, "Sampled-data supervisory control," Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont, 2009.

[15] K. Rudie, "Software for the control of discrete-event systems: A complexity study," Master's thesis, Dept. of Electrical and Computer Engineering, University of Toronto, Toronto, Ont, 1988.

[16] R. Leduc, "PLC implementation of a DES supervisor for a manufacturing testbed: An implementation perspective," Master's thesis, Dept. of Elec and Comp Eng, University of Toronto, Toronto, Ont, 1996.

[17] DESpot, "`www.cas.mcmaster.ca/~leduc/DESpot.html`. The official website for the DESpot project," 2013.