Sampled-Data Controller Implementation

Yu Wang and Ryan J. Leduc

Dept. of Computing and Software, McMaster University
email: wangy22@cas.mcmaster.ca, leduc@mcmaster.ca

***Abstract*— The setting of this paper is the implementation of timed discrete-event systems (TDES) as sampled-data (SD) controllers. An SD controller is driven by a periodic clock and sees the system as a series of inputs and outputs. On each clock edge (*tick* event), it samples its inputs, changes states, and updates its outputs.**

In this paper, we establish a formal representation of an SD controller as a Moore synchronous finite state machine (FSM). We describe how to translate a TDES supervisor to a FSM, as well as necessary properties to be able to do so. We discuss how to construct a single centralized controller as well as a set of modular controllers, and show that they will produce equivalent output.

We also discuss a flexible manufacturing system (FMS) example and present some FSM translation issues encountered, as well as the FSM version of some of the system's supervisors.

I. Introduction

In the area of Discrete-Event Systems (DES) [1], [2], [3], a lot of effort has been devoted to studying standard properties such as nonblocking and controllability in a theoretical setting. However, limited effort has been made in investigating what an implementation of a DES supervisor would be like.

A good implementation method for DES supervisors would be as *sampled-data (SD) controllers*. An SD controller is driven by a periodic clock and sees the system as a series of inputs and outputs. On each clock edge, it samples its inputs, changes state, and updates its outputs. An example of an SD controller might be a programmable logic controller (PLC) [4] or a Moore synchronous finite state machine (FSM) [5]. We are particularly interested in implementing timed DES (TDES) [6] as SD controllers.

When we are using an SD controller to manage a given system, we associate an input with each event, and an output with each controllable event. We consider an event to have occurred when its corresponding input has gone true during a given clock period. We consider a controllable event to be enabled when its corresponding output has been set true by the controller, disabled otherwise. Finally, we associate the clock edge that drives the SD controller with the TDES *tick* (τ) event.

These definitions have several ramifications. First, an SD controller does not know an event has occurred until the next clock edge, and then it has no information on the order or number of occurrences of events. The only ordering information that remains is which *sampling period* (clock period) a given event occurred in.

As an example, consider Fig. 1. We see on the third rising edge of the clock, the SD controller knows that both events *e1* and *e2* have occurred, but not which came first. This means that the SD controller can't tell the difference between the strings *e1-e2-*τ, *e2-e1-*τ, or *e1-e2-e1-*τ.

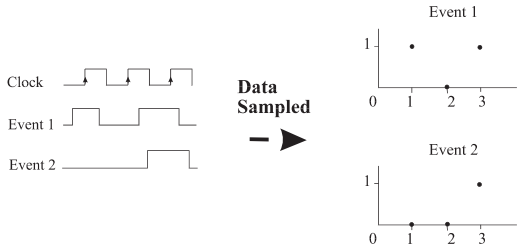


Fig. 1. Sampling Events

Second, forcing and disablement decisions can only be made immediately after the clock edge and are constant till the next clock edge. These issues raise concerns with respect to controllability, nonblocking, plant model correctness, and the SD controllers ability to determine which state the TDES system currently is in.

In [7], [8], we identified a number of existing TDES properties as well as introduced some new properties, in particular SD controllability, to address issues related to timing delay and concurrency inherent in an implementation, and to make the translation from a TDES supervisor to an SD controller easier. For a discussion on how SD controllability compares to earlier works, see [8].

In this paper, we will focus on formalizing SD controllers and defining how to translate a TDES supervisor into an SD controller. We establish a formal representation of an SD controller as a Moore synchronous finite state machine, and describe how to translate a TDES supervisor to an FSM, as well as necessary properties to be able to do so. We discuss how to construct a single centralized controller as well as a set of modular controllers, and show that they will produce equivalent output.

We will also discuss a flexible manufacturing system (FMS) example that we had earlier applied the SD controllability definition to and present some FSM translation issues encountered, as well as the FSM version of some of the system's supervisors.

In Section II, we discuss TDES preliminaries. Section III gives a quick introduction to the sampled-data setting. In Section IV, we introduce a formal representation for SD controllers as FSM, and present a centralized and modular conversion method from TDES supervisors to FSM. Next, in Section V we discuss the flexible manufacturing system example and FSM implementations.

II. Preliminaries

Below, we present a summary of the DES terminology that we use in this paper.

Let Σ be a finite set of distinct symbols (*events*), Σ^+ the set of finite sequences of events, and $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$, where ϵ is the *empty string*. 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 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. We will use the notation $\Sigma^*.\sigma$ to stand for the set of all strings $s\sigma$ for some $s \in \Sigma^*$.

For $\Omega \subseteq \Sigma$, *natural projection* $P_\Omega \colon \Sigma^* \to \Omega^*$ denotes the operation that deletes all events not in Ω from strings.

Definition 2.1: The *Nerode equivalence relation* over Σ^* mod L is defined for $s, t \in \Sigma^*$ as: $s \equiv_L t$ iff $(\forall u \in \Sigma^*) su \in L \Leftrightarrow tu \in L$.

Timed DES (TDES) [6] extends untimed DES theory by adding a new *tick* (τ) event, corresponding to the tick of a global clock. The event set of a TDES contains the *tick* event as well as other non-*tick* events called *activity events* (Σ_{act}).

A TDES automaton is represented as a 5-tuple $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$ where Q is the state set, $\Sigma = \Sigma_{act} \dot\cup \{\tau\}$ is the event set, the partial function $\delta : Q \times \Sigma \to Q$ is the transition function, q_o is the initial state, and Q_m is the set of marked states. We extend δ to $\delta : Q \times \Sigma^* \to Q$ in the natural way. The notation $\delta(q, s)!$ means the transition is defined. The *closed behavior* of $\mathbf{G}$ is defined to be $L(\mathbf{G}) := \{s \in \Sigma^* | \ \delta(q_o, s)!\}$. The *marked behavior* is defined as $L_m(\mathbf{G}) := \{s \in L(G) | \ \delta(q_o, s) \in Q_m\}$.

Definition 2.2: A DES $\mathbf{G}$ is said to be *nonblocking* if

$$\overline{L_m(\mathbf{G})} = L(\mathbf{G})$$

TDES contain *forcible* (Σ_{for}), and *prohibitable events* (Σ_{hib}). Forcible events are non-*tick* events which can be relied upon to preempt *tick*, when needed. Prohibitable events are non-*tick* events that can be disabled. The set of controllable events are $\Sigma_c = \Sigma_{hib} \dot\cup \{\tau\}$, and the uncontrollable events are $\Sigma_u = \Sigma \setminus \Sigma_c$.

The reachable state subset of DES $\mathbf{G}$ is $Q_r := \{q \in Q | \ (\exists s \in \Sigma^*)\, \delta(q_o, s) = q\}$. A DES $\mathbf{G}$ is *reachable* if $Q_r = Q$. We will always assume that a DES is reachable, has a finite state and event set, and is deterministic.

Definition 2.3: For $\mathbf{G}_i = (Q_i, \Sigma_i, \delta_i, q_{o,i}, Q_{m,i})$ $(i = 1, 2)$, we define the *synchronous product* $\mathbf{G}_1 || \mathbf{G}_2$ of the two DES as:

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

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

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

For the definitions given in this paper, we assume that our plant $\mathbf{G}$ and supervisor $\mathbf{S}$ are always combined with the synchronous product operator, thus our closed-loop system is $\mathbf{G}||\mathbf{S}$.

We now introduce an existing TDES conditions that will be useful. It ensures that TDES do not allow a *tick* event to be indefinitely preempted by activity events.

Definition 2.4: TDES $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ is *activity-loop-free (ALF)* if

$$(\forall q \in Q_r)(\forall s \in \Sigma_{act}^+) \delta(q, s) \neq q$$

We require that our closed-loop system be ALF, but it does not make sense to require our supervisors to be ALF as they may contain selfloops of prohibitable events in order to be more compact. However, selfloops provide enablement information but do not contain useful next-state information for SD controllers, so we can ignore them for translation purposes. We now introduce the definition below that will make supervisors easier to translate.

Definition 2.5: Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ be a TDES, and let $\mathbf{G}'$ be $\mathbf{G}$ with all activity event selfloops removed. $\mathbf{G}$ is *non-selfloop activity-loop-free* if $\mathbf{G}'$ is ALF.

III. Sampled-Data Setting

In this section, we introduce the sampled-data setting from [7], [8]. We will take Σ to be our system event set. For SD controllers, we make the following assumptions about our system. It is the designer's responsibility to ensure that they are met.

1) The set of prohibitable events is exactly equal to the set of forcible events for our system. This is a reasonable assumption that will greatly simplify things.
2) Enabling a prohibitable event means we should force the event during the current clock period. We only allow it to occur once per clock period.
3) We assume an event has "occurred" when its input goes true unless this occurs so close to the clock edge, it shows up in the next sampling period. In that case, it "occurs" immediately after the clock edge. This should be reflected in the system model.
4) The length of a given input pulse for an event is such that the controller will never miss it, or interpret the event as occurring in the wrong clock period.

For SD controllers, we identify a *tick* event with the clock edge that the SD controller uses for sampling and state change. This means the strings an SD controller can observe are ϵ and strings ending with a *tick*. We refer to these strings as *sampled strings*, defined as $L_{samp} = \Sigma^*.\tau \cup \{\epsilon\}$.

An SD controller changes state after each clock edge (*tick*). Its next state is determined by all the strings that can occur containing a single *tick* event at the end, since the last *tick* event. We refer to such strings as *concurrent strings*, defined as $L_{conc} = \Sigma_{act}^*.\tau \subset L_{samp}$.

For TDES supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, states reached from the initial state by sampled strings represent states in $\mathbf{S}$ that are at least partially observable. We refer to such states as *sampled states*, defined as:

$$X_{samp} := \{x \in X | x = \xi(x_o, s) \text{ for some } s \in L(\mathbf{S}) \cap L_{samp}\}$$

However, if two strings contain the exact same events but in different order and/or number, they are indistinguishable to the controller. To capture this uncertainty, we define the *occurrence* operator. It takes a string and returns the set of events (the *occurrence image*) that make up the string.

Definition 3.1: For $s \in \Sigma^*$, the *occurrence* operator, Occu : $\Sigma^* \to$ Pwr(Σ), is defined as:

$$\text{Occu}(s) := \{\sigma \in \Sigma \,|\, s \in \Sigma^*.\sigma.\Sigma^*\}$$

Clearly, we would have a problem translating a supervisor if two concurrent strings with the same occurrence image are possible at a given sampled state, but they lead to different states in $\mathbf{S}$. This would mean our controller would become nondeterministic. To ensure this doesn't happen, we require our TDES be *concurrent string deterministic*.

Definition 3.2: A TDES $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ is *concurrent string (CS) deterministic*, if

$$(\forall s \in L(\mathbf{S}) \cap L_{samp})(\forall s', s'' \in L_{conc})$$
$$[ss', ss'' \in L(\mathbf{S}) \wedge \text{Occu}(s') = \text{Occu}(s'')] \implies$$
$$[ss' \equiv_{L(\mathbf{S})} ss'' \wedge ss' \equiv_{L_m(\mathbf{S})} ss'' \wedge \xi(x_o, ss') = \xi(x_o, ss'')]$$

In Fig. 2(a), we see part of a TDES that is not CS deterministic. For this e.g., we can merge states $x'$and x'' and use the minimal version for our translation. This results in the TDES we see in Fig. 2(b), which is CS deterministic. If the two states were not equivalent, we wouldn't be able to translate the supervisor.

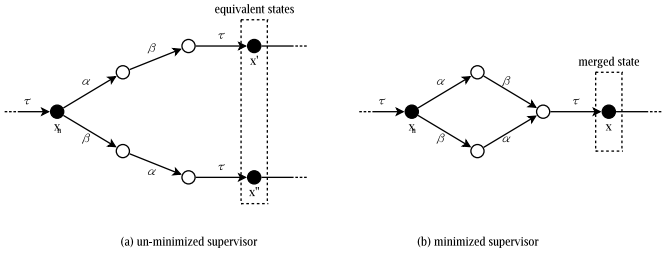


Fig. 2. CS Deterministic Example

We now can define for a given TDES a *next-sampling-state* function. This represents how a TDES will move from sampling state to sampling state via concurrent strings.

Definition 3.3: For CS deterministic TDES $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, we define the *next-sampling-state* partial function, $\Delta : X_{samp} \times \text{Pwr}(\Sigma_{act}) \rightarrow X_{samp}$, as follows. For $x \in X_{samp} \subseteq X$ and $\Sigma' \subseteq \Sigma_{act}$,

$$\Delta(x, \Sigma') := \begin{cases} \xi(x,s) & \text{if } (\exists s \in L_{conc})\xi(x,s)! \ \& \\ & \quad \text{Occu}(s) \cap \Sigma_{act} = \Sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

We use the notation $\Delta(x, \Sigma')!$ to indicate that $\Delta(x, \Sigma')$ is defined. As function Δ is only used for CS deterministic TDES, it's easy to see that it is well defined.

We define for $\mathbf{S}$ a *prohibited action* function that captures which prohibitable events are disabled at the sampled state.

Definition 3.4: For TDES supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$, the *prohibited action* function $\zeta : X_{samp} \rightarrow \text{Pwr}(\Sigma_{hib})$ is defined for $x \in X_{samp} \subseteq X$ as follows:

$$\zeta(x) := \{\sigma \in \Sigma_{hib} | \xi(x, \sigma)!\}$$

IV. Moore FSM

In this section, we introduce a formal representation for SD controllers and a translation method for TDES supervisors. We will model SD controllers as Moore synchronous finite state machine (FSM) [5]. Our choice to do so is based on the work of Leduc [9] who implemented, in an ad hoc fashion, untimed DES as FSM. Using FSM to define SD controllers is a good choice as it provides a concrete definition of the controller, yet a FSM still allows a variety of physical implementations such as a programmable logic controller (PLC) [4] program, using digital logic [5], or as a software program on a computer.

A. Formal Model

Before we give a formal definition of an SD controller, we first need to discuss some notation. We will often be discussing Boolean vectors that change periodically with respect to some clock. A Boolean vector is a vector whose individual elements can only be assigned the values of *true (1)* or *false (0)*. We say "at time k" to indicate the point of time at which k clock ticks have gone by since our starting reference point at $k = 0$. For any vector $\mathbf{v} = [v_1, v_2, ..., v_n]$, we write "$\mathbf{v}(k)$" and "$v_j(k)$" to denote the value of $\mathbf{v}$ and v_j $(j \in \{1, \ldots, n\})$ at time k. As our index k takes on new values, our vector $\mathbf{v}$ defines a sequence with respect to ticks of our clock, which we define to be $\{\mathbf{v}(k)|k = 0, 1, ...\}$, and is denoted as $\{\mathbf{v}(k)\}$ as a shorthand. When we are discussing an SD controller, we can think of $k = 0$ as representing the time when the controller has just been turned on or reset. For TDES systems, a "clock tick" corresponds to the occurrence of a *tick* event.

We define an SD controller $\mathbf{C}$ as the tupple

$$\mathbf{C} = (I, Z, Q, \Omega, \Phi, \mathbf{q}_{res})$$

where

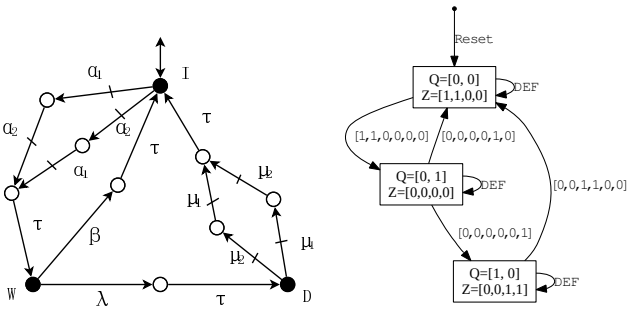
- I is the set of possible Boolean vectors that the inputs to our controller can take on. Each vector $\mathbf{i} = [i_0, i_1, .., i_{v-1}] \in I$ has v input variables. Each element of $\mathbf{i}$ corresponds to a unique activity event in our system. When an element is set to 1, this means the corresponding event has occurred at least once in the previous clock period, otherwise it is set to 0.
- Z is the set of possible Boolean vectors that the controller outputs can take on. Each vector $\mathbf{z} = [z_0, z_1, .., z_{r-1}] \in Z$ has r output variables. Each element of $\mathbf{z}$ corresponds to a unique prohibitable event in our system. When an element is set to 1, this means that corresponding event is enabled and that the controller should make the event occur before the next clock tick, where 0 means it is disabled.
- Q is the set of possible Boolean vectors that the state of our controller can take on. Each vector $\mathbf{q} = [q_0, q_1, .., q_{l-1}] \in Q$ has l state variables.
- $\mathbf{q}_{res}$ is the initial (*reset*) state for when the controller starts operating or is reset. We take $\mathbf{q}(0) = \mathbf{q}_{res}$.
- $\Omega : Q \times I \rightarrow Q$ is a next-state function which takes the current state $\mathbf{q}(k) \in Q$ and an input vector $\mathbf{i}(k+1) \in I$, and returns the next state $\mathbf{q}(k+1) = \Omega(\mathbf{q}(k), \mathbf{i}(k+1))$.
- $\Phi : Q \rightarrow Z$ is the state-to-output map. For state $\mathbf{q} \in Q$, the output $\mathbf{z} \in Z$ at this state is $\mathbf{z} = \Phi(\mathbf{q})$.

For a specific run of our controller, we would receive a specific sequence of inputs $\{\mathbf{i}(k)\}$, starting at time $k = 0$. This sequence, combined with q_{res}, and Ω, will uniquely define our current sequence of states, $\{\mathbf{q}(k)\}$. In turn, $\{\mathbf{q}(k)\}$ and Φ will uniquely define our current sequence of outputs, $\{\mathbf{z}(k)\}$.

B. Translation Method Introduction

Informally, to convert TDES $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ into SD controller $\mathbf{C} = (I, Z, Q, \Omega, \Phi, \mathbf{q}_{res})$, we take the sampled states of $\mathbf{S}$ as the states of $\mathbf{C}$. The initial (*reset*) state of $\mathbf{C}$ would be the initial state of $\mathbf{S}$. We then determine which concurrent strings are possible from a given sampled state. The occurrence image of these concurrent strings would then define our next-state conditions. For state $\mathbf{q}$ of $\mathbf{C}$, an output (enablement of some $\sigma \in \Sigma_{hib}$) is set to *true* if the corresponding event is possible at the corresponding sampled state x in $\mathbf{S}$, else set to *false*.

As an example, consider the TDES shown in Fig. 3(a) where arrows with slashes indicate prohibitable events. We see that its sampled states are I (initial state), W and D (only other states with incoming *tick* event). We thus equate the states of our FSM in 3(b) to $\mathbf{q}_{res} = [0,0] = I$, $[0,1] = W$ and $[1,0] = D$.



(a) Original TDES (b) FSM Translation

Fig. 3. FSM Translation Example

For our FSM, our ordering for the input variables is $\mathbf{i} = [\alpha_1, \alpha_2, \mu_1, \mu_2, \beta, \lambda]$ and for our outputs it is $\mathbf{z} = [\alpha_1, \alpha_2, \mu_1, \mu_2]$. Our outputs for each state of the FSM is determined by the prohibitable events possible at the corresponding sampled state of the TDES. As only α_1 and α_2 are possible at state I, only these outputs are set to 1 at state $[0,0]$. Similarly, all outputs are set to 0 at state $[0,1]$ and only μ_1 and μ_2 outputs are set to 1 at state $[1,0]$.

Examining state I, we see that the only concurrent strings leaving it are $\alpha_1\alpha_2\tau$ and $\alpha_2\alpha_1\tau$. They have the same concurrent image $\{\alpha_1, \alpha_2, \tau\}$ and both take us to state W. Our next-state condition is thus when only α_1 and α_2 have occurred, we go to state $[0,1]$ as shown in 3(b). As Ω is a total function and ξ is a partial function, we usually have to add a **DEF**, or default transition, to cover input combinations that we have not explicitly specified (i.e. it matches all remaining unspecified input combinations). We determine the next-state conditions for the remaining sampled states in a similar fashion.

The operation of our FSM is that at system reset, it starts operating at state $\mathbf{q}_{res} = [0,0]$ with its outputs set to enable only α_1 and α_2. At each clock tick, it samples its inputs creating a new input vector, **i**, which is matched to the current state's next-state conditions to determine its new state. It then changes to the new state, updates its outputs, and then waits for the next clock tick. For an example of how to implement an FSM on a programmable logic controller, see [9].

To be able to translate a TDES supervisor into an SD controller, we only require that it be CS deterministic to ensure that the resulting FSM will be deterministic. In practice, we also require that the TDES be non-selfloop ALF as a design aid. This makes it more likely that the TDES be CS deterministic and typically makes the translation easier and more compact as we will see in Section V.

Although the CS deterministic property is a sufficient condition to do the translation, it is not sufficient to ensure that the resultant controller applied to the plant will result in the desired enablement, forcing, and nonblocking behavior. As discussed in [7], [8], this requires that our supervisor $\mathbf{S}$ be SD controllable for our plant $\mathbf{G}$, $\mathbf{G}$ have proper time behavior, $\mathbf{S}$-singular prohibitable behavior, that $\mathbf{G}$ be complete for $\mathbf{S}$, and that our closed-loop system be ALF and nonblocking. Please see [7], [8] for an explanation of these properties and a discussion of the results.

C. Event Mapping Functions

As we will often be discussing vectors whose elements refer to specific events in Σ_{act}, we need a way to map events to a vector's elements and vice versa. Let $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ be the TDES plant to be controlled and let $\mathbf{S} = (X, \Sigma_{\mathbf{S}}, \xi, x_o, X_m)$ be a CS deterministic TDES supervisor for $\mathbf{G}$. Let $\mathbf{C} = (I, Z, Q, \Omega, \Phi, \mathbf{q}_{res})$ be the SD controller for $\mathbf{S}$. We will take Σ to be our system event set with $\Sigma_{\mathbf{S}} \subseteq \Sigma$.

As we could have multiple controllers, each using their own event orderings, we first need define a default bijective map, γ_g, between our activity event set and a default index set that we will use for labeling the events. To simplify things, we will require the event mapping functions for each controller to respect the event ordering imposed by γ_g. This means that γ_g will induce a single way to define the various mapping functions for our controller.

Definition 4.1: Let bijective map $\gamma_g : \Sigma_{act} \rightarrow \{0, \ldots, |\Sigma_{act}|-1\}$ be the *canonical event mapping function* for Σ_{act}.

We now define input and output mapping functions for our controller. Each function will map events to the index value for the corresponding variable in the specified vector (I or Z).

Definition 4.2: The *input event mapping function* for $\mathbf{C}$ is defined to be a bijective map $\gamma : \Sigma_{\mathbf{S}} \cap \Sigma_{act} \rightarrow \{0, 1, \ldots, v-1\}$ with $v = |\Sigma_{\mathbf{S}} \cap \Sigma_{act}|$ such that

$$(\forall \sigma_1, \sigma_2 \in \Sigma_{\mathbf{S}} \cap \Sigma_{act}) \gamma_g(\sigma_1) < \gamma_g(\sigma_2) \implies \gamma(\sigma_1) < \gamma(\sigma_2)$$

Definition 4.3: The *output event mapping function* for $\mathbf{C}$ is defined to be a bijective map $\eta : \Sigma_{\mathbf{S}} \cap \Sigma_{hib} \rightarrow \{0, 1, .., r-1\}$ with $r = |\Sigma_{\mathbf{S}} \cap \Sigma_{hib}|$ such that

$$(\forall \sigma_1, \sigma_2 \in \Sigma_{\mathbf{S}} \cap \Sigma_{hib}) \gamma_g(\sigma_1) < \gamma_g(\sigma_2) \implies \eta(\sigma_1) < \eta(\sigma_2)$$

As these functions are bijective, their inverse functions always exist. We can thus use their inverse functions to map an index value to its corresponding activity event.

D. Centralized Translation Method

We will now discuss how to translate a TDES supervisor into a single centralized controller. Let TDES $\mathbf{S} =$

$(X, \Sigma, \xi, x_o, X_m)$ be CS deterministic. To translate $\mathbf{S}$ into a controller $\mathbf{C} = (I, Z, Q, \Omega, \Phi, \mathbf{q}_{res})$, we need to determine values for the members of the tupple.

We will start with I, Z, and Q. As they represent all possible assignments of Boolean vectors, we first need to define the size (number of elements/variables) of each vector. The size of each input vector $\mathbf{i} \in I$ is defined to be $v = |\Sigma_{act}|$. The size of each output vector $\mathbf{z} \in Z$ is defined to be $r = |\Sigma_{hib}|$.

We now specify Q. We need to define the size of the state vectors, l, so that a state vector is large enough to encode a value for each sampled state $x \in X_{samp} \subseteq X$. We thus choose l to satisfy $2^{l-1} < |X_{samp}| \leq 2^l$.

To complete our definition of I, Z, and Q, we now need to associate activity events with individual input variables, prohibitable events with individual output variables, and sampled states with state assignments of Q.

Definition 4.4: Let γ be the input event mapping function for controller $\mathbf{C}$. We define for $\mathbf{C}$ the *input set mapping* bijective function, $\Gamma_I : Pwr(\Sigma_{act}) \rightarrow I$, as follows. For $\Sigma' \subseteq \Sigma_{act}$, we have $\Gamma_I(\Sigma') = [i_0, i_1, .., i_{v-1}]$ such that for $j = 0, 1, .., v-1$,

$$i_j := \begin{cases} 1 & \text{if } (\exists \sigma \in \Sigma')\gamma(\sigma) = j \\ 0 & \text{otherwise} \end{cases}$$

Definition 4.5: Let η be the output event mapping function for controller $\mathbf{C}$. We define for $\mathbf{C}$ the *output set mapping* bijective function, $\Gamma_Z : \text{Pwr}(\Sigma_{hib}) \rightarrow Z$, as follows. For $\Sigma' \subseteq \Sigma_{hib}$, we have $\Gamma_Z(\Sigma') = [z_0, z_1, .., z_{r-1}]$ such that for $j = 0, 1, .., r-1$,

$$z_j := \begin{cases} 1 & \text{if } (\exists \sigma \in \Sigma')\eta(\sigma) = j \\ 0 & \text{otherwise} \end{cases}$$

Definition 4.6: We define for controller $\mathbf{C}$ the *state set mapping* function, $\Lambda : X_{samp} \rightarrow Q$, to be an arbitrary injective function of the designer's choice.

We can now define the reset state for $\mathbf{C}$ as $\mathbf{q}_{res} = \Lambda(x_o)$. Next, we define the next-state and state-to-output maps.

Definition 4.7: Let Δ be the next-sampling-state partial function for supervisor $\mathbf{S}$. For state $\mathbf{q} \in Q$ and input $\mathbf{i} \in I$, the *next-state* function, Ω, is defined to be

$$\Omega(\mathbf{q}, \mathbf{i}) := \begin{cases} \Lambda(\Delta(x, \Gamma_I^{-1}(\mathbf{i}))) \text{ if } (\exists x \in X_{samp})\mathbf{q} = \Lambda(x) \ \& \\ \qquad\qquad \Delta(x, \Gamma_I^{-1}(\mathbf{i}))! \\ \text{arbitrary} \qquad\qquad \text{otherwise} \end{cases}$$

Definition 4.8: Let ζ be the *prohibited action* function for supervisor $\mathbf{S}$. For state $\mathbf{q} \in Q$, the *state-to-output map* Φ is defined to be

$$\Phi(\mathbf{q}) := \begin{cases} \Gamma_Z(\zeta(x)) & \text{if } (\exists x \in X_{samp})\mathbf{q} = \Lambda(x) \\ \Gamma_Z(\emptyset) & \text{otherwise} \end{cases}$$

We now have completely defined our controller $\mathbf{C}$ for TDES supervisor $\mathbf{S}$. As long as $\mathbf{S}$ is CS deterministic, then its next-sampling-state function Δ will be well defined, allowing us to do the translation. Based on the above definitions, it is easy to see that for $\Sigma' \subseteq \Sigma_{act}$ and $x \in X_{samp}$, if $\Delta(x, \Sigma')!$, then the diagram in Fig. 4 commutes.

So far we have assumed our supervisor's event set is Σ. If our supervisor is defined over a subset $\Sigma_{\mathbf{S}} \subset \Sigma$, we would simply add to every state of our supervisor selfloops for each $\sigma \in \Sigma \setminus \Sigma_{\mathbf{S}}$ and use this new supervisor for the translation.

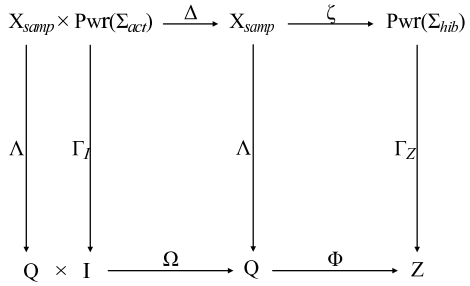


Fig. 4. Centralized Control Equivalence Diagram

E. Output Equivalence

Before we define a modular translation method, we first need to consider how to determine if two different controllers would produce equivalent output (i.e. enablement information) for the same input sequence. The problem is that the controllers could be defined over different sets of input events and might use different event ordering for their vectors.

To handle different input requirements, we will assume that we have a system input vector, $\mathbf{i}_g$, containing all activity events and ordered with respect to the canonical event mapping function, γ_g. We call $\{\mathbf{i}_g(k)\}$ a *canonical input sequence* and $\mathbf{i}_g \in \{\mathbf{i}_g(k)\}$ a *canonical input vector*. We can then map each $\mathbf{i}_g$ to a corresponding input vector for each controller using γ_g and the controllers own output event mapping function, γ. For example, we can map $\mathbf{i}_g = [i_{g,0}, i_{g,1}, .., i_{g,v_g-1}]$ $(v_g = |\Sigma_{act}|)$ to $\mathbf{i} = [i_0, i_1, .., i_{v-1}]$ by setting $i_l = i_{g,l'}$ for $l = 0, \ldots, v-1$, with $l' = \gamma_g((\gamma^{-1}(l)))$.

In particular, we are interested in comparing two controllers $\mathbf{C}_1$ and $\mathbf{C}_2$ that have been defined relative to the same CS deterministic supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$. We are thus only interested in determining if they generate the same output with respect to input sequences that represent valid input strings to $\mathbf{S}$ (i.e. $s \in L(\mathbf{S}) \cap L_{samp}$).

Definition 4.9: A canonical input sequence $\{i_g(k)\}$ is *input valid* for $\mathbf{S}$, if

$(\forall k' \in \{1, 2, \ldots\})(\exists s_1, s_2, \ldots, s_{k'} \in L_{conc})$
$[s_1 s_2 \ldots s_{k'} \in L(\mathbf{S})] \wedge [(\forall n \in \{1, 2, \ldots, k'\})(\forall \sigma \in \Sigma_{act})$
$i_{g,\gamma_g(\sigma)}(n) = 1$ iff $\sigma \in \text{Occu}(s_n)]$

Essentially in the above definition, we are requiring the sequence $\{i_g(k)\}$ to correspond to a sequence of concurrent strings that supervisor $\mathbf{S}$ will accept.

Definition 4.10: Let $\mathbf{C}_j = (I_j, Z_j, Q_j, \Omega_j, \Phi_j, \mathbf{q}_{res,j})$ be an SD controller with output mapping function η_j, $j = 1, 2$. We say $\mathbf{C}_1$ and $\mathbf{C}_2$ are *output equivalent with respect to* $\mathbf{S}$ if for any canonical input sequence $\{\mathbf{i}_g(k)\}$ that is input valid for $\mathbf{S}$, and induced outputs $\mathbf{z}_j(k') = [z_{j,1}(k'), z_{j,2}(k'), .., z_{j,r_j}(k')] \in Z_j$ $(j = 1, 2)$ at time $k' = \{0, 1, 2, \ldots\}$, the follow conditions are satisfied:

1) $r_1 = r_2$
2) $(\forall 0 \leq i < r_1)\eta_1^{-1}(i) = \eta_2^{-1}(i)$
3) $(\forall k' \in \{0, 1, ..\})\mathbf{z}_1(k') = \mathbf{z}_2(k')$

Points 1 and 2 require the outputs of the two controllers to be of the same size, and event ordering. Point 3 requires that one controller enables a prohibitable event if and only if the other does, for the same value of k'.

F. Modular Translation Method

For large systems, we typically define modular supervisors rather than a single centralised supervisor. We would like to translate each supervisor into their own modular controller, and then combine their outputs to produce the equivalent enablement information of a centralized controller.

Let TDES $\mathbf{S} = \mathbf{S}_1||\mathbf{S}_2||..||\mathbf{S}_n$ be a CS deterministic supervisor where each modular supervisor $\mathbf{S}_j = (X_j, \Sigma_j, \xi_j, x_{o,j}, X_{m,j})$, $j = 1, 2, .., n$, is also CS deterministic. Let $\Sigma_j = \Sigma_{act,j} \dot{\cup} \{\tau\}$, $\Sigma_{hib,j} := \Sigma_{hib} \cap \Sigma_{act,j}$, and $\Sigma = \bigcup_{l \in \{1,2,..,n\}} \Sigma_l$.

For the following discussion, we will define the logical AND of vectors $\mathbf{u} = [u_1, u_2, .., u_m]$ and $\mathbf{v} = [v_1, v_2, .., v_m]$ as $\mathbf{u} \wedge \mathbf{v} = [u_1 \wedge v_1, u_2 \wedge v_2, .., u_m \wedge v_m]$. We define the concatenation of vectors $\mathbf{u} = [u_1, u_2, .., u_{m_1}]$ and $\mathbf{v} = [v_1, v_2, .., v_{m_2}]$ as $\mathbf{uv} = [u_1, u_2, .., u_{m_1}, v_1, v_2, .., v_{m_2}]$.

Definition 4.11: For $j = 1, 2, .., n$, the j^{th} modular translation of CS deterministic supervisor $\mathbf{S}_j$ to modular controller $\mathbf{C}_j = (I_j, Z_j, Q_j, \Omega_j, \Phi_j, \mathbf{q}_{res,j})$ is performed using the method defined in Section IV-D after replacing Σ_{act} by $\Sigma_{act,j}$, and Σ_{hib} with $\Sigma_{hib,j}$ in the definitions.

To define how we combine the individual outputs of the modular controllers together, we need to define the composite controller of our n modular controllers.

Definition 4.12: The composite controller for $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n$, $\mathbf{C} = (I, Z, Q, \Omega, \Phi, \mathbf{q}_{res}) = \mathbf{comp}(\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_n)$, is defined as follows:

- The size of each input vector $\mathbf{i} \in I$ is defined to be $v = |\Sigma_{act}|$. The size of each output vector $\mathbf{z} \in Z$ is defined to be $r = |\Sigma_{hib}|$. Let γ to be the input event mapping function for $\mathbf{C}$ and η to be the output event mapping function.
- The number of state variables for vectors $\mathbf{q} \in Q$ is defined to be $l = \sum_{j=1}^{n} l_j$, where l_j is the number of state variables for Q_j. The reset state is defined to be $\mathbf{q}_{res} = \mathbf{q}_{res,1}\mathbf{q}_{res,2} \cdots \mathbf{q}_{res,n}$.
- The next-state function Ω is defined such that, for $\mathbf{q}(k) = \mathbf{q}_1(k)\mathbf{q}_2(k)..\mathbf{q}_n(k) \in Q$ and $\mathbf{i}(k+1) \in I$,
  $$\begin{aligned}\mathbf{q}(k+1) &= \Omega(\mathbf{q}(k), \mathbf{i}(k+1)) \\ &= \Omega_1(\mathbf{q}_1(k), \mathbf{i}_1(k+1)) \ldots \Omega_n(\mathbf{q}_n(k), \mathbf{i}_n(k+1))\end{aligned}$$
  where input vector $\mathbf{i}(k+1)$ in canonical form with respect to γ_g, was mapped to input vector $\mathbf{i}_j(k+1)$ ($j = 1, \ldots, n$) as described in Section IV-E.
- The state-to-output map Φ is defined as follows. For $\mathbf{q} = \mathbf{q}_1\mathbf{q}_2..\mathbf{q}_n \in Q$, let $\mathbf{z}_j = \Phi_j(\mathbf{q}_j) = [z_{j,0}, z_{j,1}, \ldots, z_{j,r_j-1}] \in Z_j$, $j = 1, \ldots, n$. For each $\mathbf{z}_j$, we translate it to $\mathbf{z}'_j = [z'_{j,0}, z'_{j,1}, .., z'_{j,r-1}] \in Z$ such that,
  $$(\forall \sigma \in \Sigma_{hib}) z'_{j,\eta(\sigma)} = \begin{cases} z_{j,\eta_j(\sigma)} & \text{if } \sigma \in \Sigma_{hib,j} \\ 1 & \text{otherwise} \end{cases}$$
  We can now define:
  $$\Phi(\mathbf{q}) = \bigwedge_{j \in \{1,2,..,n\}} \mathbf{z}'_j$$

The following theorem essentially states that the enablement information of the composite controller is equivalent to that of the centralized controller.

Theorem 4.1: Let CS deterministic supervisors $\mathbf{S} = \mathbf{S}_1||\mathbf{S}_2||..||\mathbf{S}_n$ and $\mathbf{S}_j$ ($j = 1, 2, .., n$) be defined as above. Let $\mathbf{C}_j$ be the modular controller for $\mathbf{S}_j$, $\mathbf{C}' = \mathbf{comp}(\mathbf{C}_1, \mathbf{C}_2, .., \mathbf{C}_n)$ be the composite controller, and $\mathbf{C}$ be the centralized controller translated from $\mathbf{S}$;

then $\mathbf{C}$ and $\mathbf{C}'$ are output equivalent with respect to $\mathbf{S}$.

Proof: See proof in [7]. ■

V. FMS Example

In this section we discuss an example based on the untimed flexible manufacturing system (FMS) from Hill [10]. The system, shown in Fig. 5, is composed of six plant components and five one slot buffers. We will treat the buffers as specifications, requiring that they do not overflow or underflow. The flow of material is illustrated in Fig. 5.

Table I below shows an explanation of the numeric event labels used. Events labeled as numbers are directly from the Hill example, where even numbers represent uncontrollable events. For TDES, marked states are indicated by a gray circle, and initial states have a thick outline.

TABLE I
MEANING OF EVENT LABELS

Label	Meaning	Label	Meaning
921	Part enters system	*922*	Part enters B2
933	Robot takes from B2	*934*	Robot to B4
937	B4 to Robot for B6	*939*	B4 to Robot for B7
938	Robot to B6	*930*	Robot to B7
951	B4 to Lathe (A)	*953*	B4 to Lathe (B)
952	Lathe to B4 (A)	*954*	Lathe to B4 (B)
971	B7 to Con3	*974*	Con3 to B7
972	Con3 to B8	*973*	B8 to Con3
981	B8 to PM	*982*	PM to B8
961	Initialize AM	*963*	B6 to AM
965	B7 to AM	*966*	Finished from B7
964	Finished from B6		

The plant components consist of two conveyors (**Con2** and **Con3**), a handling robot (**Robot**), a lathe that can produce two different parts (A and B), a painting machine (**PM**), and a finishing machine (**AM**). In total, we have 10 plant TDES and 15 modular supervisor TDES. Wang's thesis [7] provides complete details for each TDES, as well as a discussion on how to apply the new SD controllability and related definitions to the example. In this paper, we will focus on translating the supervisors into SD controllers.

When we examine the supervisors from [7], we find that supervisors **B4**, **B4Path**, **B6**, and **B7**, shown at the top of Fig. 6, are neither non-selfloop ALF nor CS deterministic.

For example, consider state 0 of supervisor **B4**. We see we can do a *934-tick* sequence, a $\{$*934-951*$\}$*-tick* sequence, and a *934-951-934-tick* sequence, among others. Not only would our controller be nondeterministic, but these sequences provide us with numerous next-state conditions, many of which are not possible in the system. Examining the plants and supervisors in [7], we see that there will always be a

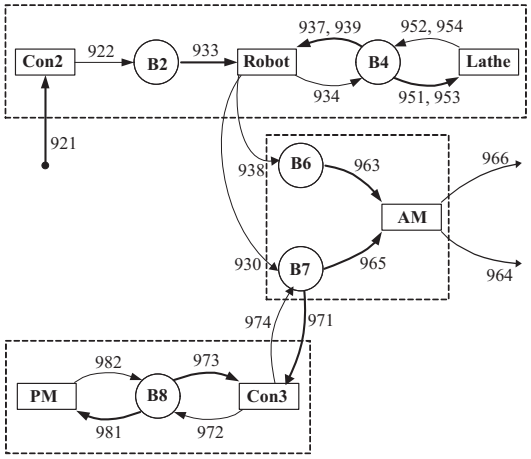


Fig. 5. Flexible Manufacturing System Overview

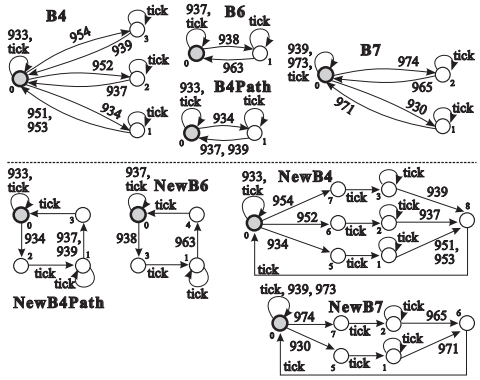


Fig. 6. Corrected TDES Supervisors

tick event between events *934* and *951*, so we can add this to our supervisor without restricting the closed-loop behavior of our system. Making similar observations for the other non-selfloop activity loops, we arrive at supervisor **NewB4**, shown in Fig. 6, which is now non-selfloop ALF and CS deterministic. Similarly, we derive replacement supervisors **NewB4Path**, **NewB6**, and **NewB7**, also shown in Fig. 6.

We now convert these new supervisors into modular controllers using the method described in Section IV-F. The results are shown in Figure 7. To see the SD controllers for the remaining supervisors for our system, please refer to [7].

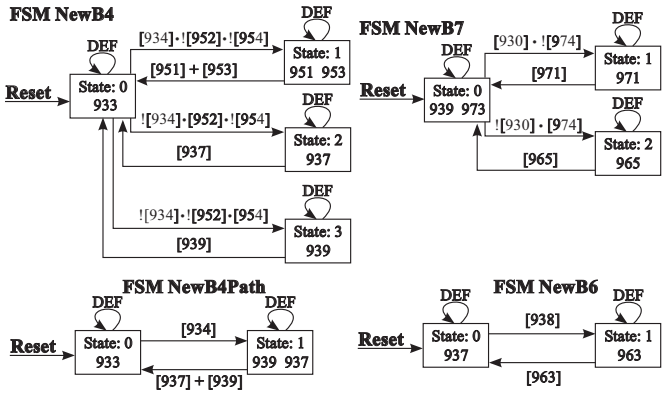


Fig. 7. FSM for New Supervisors

The controllers shown use a more compact notation than that used in Fig. 3(b). At each state in the FSM, we only list prohibitable events whose outputs are set to true. For next-state conditions, we use Boolean equations to express input conditions and we simplify them to only include input events possible at that state. This is done to make the FSM more compact and human readable.

For example, at state 0 of FSM NewB7 we see the next-state condition $[930]\cdot![974]$ which means event 930 occurred in the last sampling period, but event 974 did not. It will match any input vector for which this is true. Here we are representing logical AND as "$\cdot$," and logical negation as "!". We also represent logical OR as "$+$."

To keep the diagrams for FSM NewB4Path and FSM NewB4 simple, we took a couple of shortcuts. This was done strictly for readability purposes, and would not be done with an automated conversion process. Technically, the next-state condition from state 1 to state 0 of FSM NewB4Path should be $([937]\cdot![939])+(![937]\cdot[939])$. However, examining the plant and other supervisors discussed in [7] we see that events 937 and 939 can never occur in the same clock period so we can safely simplify this condition to $[937]+[939]$.

VI. Conclusions

In this paper, we establish a formal representation of an SD controller as a Moore synchronous finite state machine (FSM), and describe how to translate a TDES supervisor to a FSM, as well as necessary properties to be able to do so. We discuss how to construct a single centralized controller as well as a set of modular controllers, and show that they will produce equivalent output. Implementing modular supervisors as a set of modular controllers provides a compact implementation method.

We also discuss a flexible manufacturing system (FMS) example that we had earlier applied the SD controllability definition to and present some FSM translation issues encountered, as well as the FSM version of some of the system's supervisors.

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 Elec and Comp Eng, University of Toronto, July 2008.

[4] W. Bolton, *Programmable Logic Controllers*, 4th ed. Elsevier, 2006.

[5] S. Brown and Z. Vranesic, *Fundamentals of Digital Logic with VHDL Design*, 3rd ed. McGraw Hill Higher Education, 7 2008.

[6] B. Brandin and W. M. Wonham, "Supervisory control of timed discrete-event systems," *IEEE Trans. on Auto Cont*, pp. 329–342, Feb 1994.

[7] Y. Wang, "Sampled-data supervisory control," Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont, 2009, [ONLINE] http://www.cas.mcmaster.ca/˜leduc/.

[8] R. J. Leduc and Y. Wang, "Sampled-data supervisory control," in *Proc. of WODES 2010*, Berlin, Germany, Aug. 2010, pp. 353–359.

[9] 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.

[10] R. C. Hill, "Modular verification and supervisory controller design for discrete-event systems using abstraction and incremental construction," Ph.D. dissertation, Dept. of Mech Eng, University of Michigan, 2008.