Sampled-Data Supervisory Control

Ryan J. Leduc * **Yu Wang** **

* *Department of Computing and Software, McMaster University, Hamilton, Canada, (e-mail: leduc@mcmaster.ca)*
** *Department of Computing and Software, McMaster University, Hamilton, Canada, (e-mail: wangy22@mcmaster.ca)*

Abstract:
This paper focuses on issues related to implementing timed discrete-event systems (TDES) supervisors, and the concurrency and timing delay issues involved. In particular, we examine issues related to implementing 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. We extend TDES controllability to a new definition, SD controllability, which captures several new properties that are useful in dealing with concurrency issues, as well as makes it easier to translate a TDES supervisor into an SD controller. We present controllability and nonblocking results for SD controllers. Finally, we apply our method to a small manufacturing system from the literature.

Keywords: Discrete-event systems (DES), Timed DES, implementation, concurrency.

1. INTRODUCTION

In the area of Discrete-Event Systems (DES) [Ramadge and Wonham(1987)], [Wonham and Ramadge(1987)], [Wonham(2008)], 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, whether we can guarantee that it will retain the controllability and nonblocking properties of the theoretical supervisor, and how to handle timing delay and concurrency issues inherent in an implementation.

1.1 Sampled-Data Controllers

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) [Bolton(2006)] or a Moore synchronous finite state machine (FSM) [Brown and Vranesic(2008)]. For simplicity, we will assume inputs and outputs of an SD controller can take the values of *true* or *false*. In this paper, we will focus on the timing and concurrency issues involved in implementing timed DES (TDES) [Brandin and Wonham(1994)] 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.

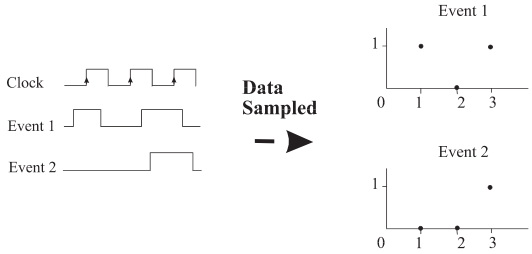

Fig. 1. Sampling Events

Concurrency and timing issues arise when implementing TDES as SD controllers as TDES assume that events occur in an interleaving fashion (i.e. we can always determine event ordering), we know immediately when events occur, and enablement and forcing occur immediately (i.e. no communication delay). These assumptions are false in general for SD controllers. Also, due to variations in how an SD controller is designed, the occurrence of a

⋆ Initial work on this topic was begun during Leduc's doctoral thesis under Prof. W.M. Wonham, and in collaboration with Dr. Bertil A. Brandin.

forced event in a specified clock period will vary and could possibly occur in any order relative to the other events occurring in the same clock period. These have ramifications with respect to controllability, plant model correctness, and the SD controllers ability to determine which state the TDES system currently is in.

Finally, if an SD controller is forcing multiple events (say α and β) to occur in the same clock period, these events may only actually occur (due to design or timing constraints) in a specific order (say $\alpha\beta$ only), even though the TDES model says they can occur in multiple orderings (say $\beta\alpha$ as well). This means a TDES could be nonblocking, but its SD controller implementation could block.

In this paper, we will develop a new property for TDES systems that will address the above issues, as well as make our TDES supervisor more consistent with SD controllers, making them easy to translate.

In Section 2, we discuss TDES preliminaries, including introducing some TDES related properties that we will need. Section 3 introduces the sampled-data setting, including a new property called SD controllability. In Section 4, we present controllability and nonblocking results for SD controllers. In Section 5 we apply our method to a small manufacturing example from the literature. We finish with conclusions and future work.

1.2 Related Work

In the sampled-data setting, if the same event occurs once or multiple times in the same sampling period, an SD controller will not be able to detect the difference. In [Basile and Chiacchio(2007)], the authors require that the system have the property that an event cannot be generated more than once during a sampling period. The paper also discussed the loss of ordering information when events occur in the same sampling period. To handle these timing related issues, the author adds a dispatcher to the existing supervisor to solve the problems that could occur when event ordering cannot be ignored. The model is implemented based on Petri Nets and an algorithm to translate the Petri Net implementation into a programming language is provided.

In [Leduc(1996)], the author investigated implementing DES as Moore finite state machines (FSM) and created an implementation by hand for his example. The idea of implementing TDES as SD controllers was motivated by this work. [Nourelfath and Niel(2004)] discuss translating DES into PLC programs. They first convert automata into the Grafcet language, which describes the specification of logic controllers. They then translate the Grafcet language into a PLC program.

In [Dragert et al.(2008)Dragert, Dingel, and Rudie], DES theory is used as a tool to assist in the development of concurrent software. The authors describe an approach to generate program source code for currency control from specifications. The approach takes portions of code that do not contain concurrency control code, but instead markup describing events that are relevant to the specification. A supervisor is generated from this information and then converted into concurrency control code.

A real world application of DES supervisory control is given in [Giua and Seatzu(2008)], where Petri Nets are used to model railway networks.

However, even if the DES supervisor is nonblocking for the DES plant does not mean that the controller implementation is nonblocking as well. To ensure a controller is nonblocking, [Malik(2003)] studied several different methods for implementing controllers. The author suggested conditions to be satisfied for the implemented controllers to be nonblocking.

Another practical issue for implementing controllers is communication. In [Schmidt and Schmidt(2008)], the authors study the communication between modular and decentralized supervisors on a switched network. In [Xu and Kumar(2008)], the authors resolve communication issues by introducing an asynchronous implementation.

2. PRELIMINARIES

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

2.1 Strings and Languages

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 $\mathrm{Pwr}(\Sigma)$ denote the set of all possible subsets of Σ.

For $\Omega \subseteq \Sigma$, *natural projection* $P_\Omega \colon \Sigma^* \to \Omega^*$ denotes the operation that deletes all events not in Ω from strings. For language $L \subseteq \Sigma^*$, the eligibility operator $\mathrm{Elig}_L : \Sigma^* \to \mathrm{Pwr}(\Sigma)$ is given by $\mathrm{Elig}_L(s) := \{\sigma \in \Sigma \,|\, s\sigma \in L\}$ for $s \in \Sigma^*$.

Definition 1. The *Nerode equivalence relation* over Σ^* mod L is defined for $s, t \in \Sigma^*$ as: $s \equiv_L t$ iff $(\forall u \in \Sigma^*) su \in L \Leftrightarrow tu \in L$.

2.2 Timed DES

Timed DES (TDES) [Brandin and Wonham(1994)] 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 marker 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. 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 $\Sigma_u = \Sigma - \Sigma_c$.

The reachable state subset of DES $\mathbf{G}$, denoted Q_r, 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 3. For $\mathbf{G}_i = (Q_i, \Sigma, \delta_i, q_{o,i}, Q_{m,i})$ $(i = 1, 2)$, we define the *product* of the two DES as:

$\mathbf{G_1} \times \mathbf{G_2} := (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_{o,1}, q_{o,2}), Q_{m,1} \times Q_{m,2})$, where $\delta_1 \times \delta_2 : Q_1 \times Q_2 \times \Sigma \to Q_1 \times Q_2$ is given by $(\delta_1 \times \delta_2)((q_1, q_2), \sigma) := (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$, whenever $\delta_1(q_1, \sigma)!$ and $\delta_2(q_2, \sigma)!$.

We note that $L(\mathbf{G_1} \times \mathbf{G_2}) = L(\mathbf{G_1}) \cap L(\mathbf{G_2})$ and $L_m(\mathbf{G_1} \times \mathbf{G_2}) = L_m(\mathbf{G_1}) \cap L_m(\mathbf{G_2})$.

In this paper, we assume all DES are combined with the product DES operator. For plant $\mathbf{G}$ and supervisor $\mathbf{S}$, our closed-loop system is $\mathbf{G} \times \mathbf{S}$.

Definition 4. Language $K \subseteq L(\mathbf{G})$ is *controllable* with respect to $\mathbf{G}$ if for all $s \in \overline{K}$,

$$\mathrm{Elig}_{\overline{K}}(s) \supseteq \begin{cases} \mathrm{Elig}_{L(\mathbf{G})}(s) \cap (\Sigma_u \cup \{\tau\}) & \text{if } \mathrm{Elig}_{\overline{K}}(s) \cap \Sigma_{for} = \emptyset \\ \mathrm{Elig}_{L(\mathbf{G})}(s) \cap \Sigma_u & \text{if } \mathrm{Elig}_{\overline{K}}(s) \cap \Sigma_{for} \neq \emptyset \end{cases}$$

Definition 5. A *supervisory control* for $\mathbf{G}$ is a map $V : L(\mathbf{G}) \to \mathrm{Pwr}(\Sigma)$, such that, for each $s \in L(\mathbf{G})$,

$$V(s) \supseteq \begin{cases} \Sigma_u \cup (\{\tau\} \cap \mathrm{Elig}_{L(\mathbf{G})}(s)) & \text{if } V(s) \cap \mathrm{Elig}_{L(\mathbf{G})}(s) \cap \Sigma_{hib} = \emptyset \\ \Sigma_u & \text{if } V(s) \cap \mathrm{Elig}_{L(\mathbf{G})}(s) \cap \Sigma_{hib} \neq \emptyset \end{cases}$$

Definition 6. We write $V/\mathbf{G}$ to represent $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ under the supervision of V. The *closed behavior* of $V/\mathbf{G}$ is defined to be $L(V/\mathbf{G}) \subseteq L(\mathbf{G})$ such that

(1) $\epsilon \in L(V/\mathbf{G})$;
(2) if $s \in L(V/\mathbf{G})$, $\sigma \in V(s)$ and $s\sigma \in L(\mathbf{G})$, then $s\sigma \in L(V/\mathbf{G})$;
(3) no other strings are in $L(V/\mathbf{G})$.

2.3 TDES Properties

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

Definition 7. 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 will also require that our plant TDES have *proper time behavior*, as defined by Kai Wong et al. [Wong and Wonham(1996)].

Definition 8. TDES $\mathbf{G}$ has a *proper time behavior* if

$$(\forall q \in Q_r)(\exists \sigma \in \Sigma_u \cup \{\tau\})\ \delta(q, \sigma)!$$

If the plant is also ALF and has a finite statespace, then we can always reach a state where a *tick* is possible after at most a finite number of activity events.

Controllable events often are part of the supervisor's implementation and can occur when we want them to [Balemi(1994)]. However, a plant might be modeled more restrictively in order to make the system easier to model or understand. We have modified the definition to only use prohibitable events.

Definition 9. Let TDES $\mathbf{G}$ be a plant and TDES $\mathbf{S}$ be a supervisor. $\mathbf{G}$ is *complete* for $\mathbf{S}$ if

$$(\forall s \in L(\mathbf{G}) \cap L(\mathbf{S}))(\forall \sigma \in \Sigma_{hib}) s\sigma \in L(\mathbf{S}) \implies s\sigma \in L(\mathbf{G})$$

3. SAMPLED-DATA CONTROLLERS

In this section, we will examine the behavior of SD controllers in more detail.

3.1 Assumptions

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. This is essentially a modeling issue.
(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) Our SD controllers will be implemented centrally with a common clock, such that they all sample inputs, and update outputs at the same time.
(4) 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.
(5) When we force an event in a given sampling period, it occurs sometime during that clock period.
(6) 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.

Applicable systems should not find Assumptions 1, 2, 5, and 6 very restrictive. Assumptions 3 and 4 partially address timing delay issues and likely will be removed in future work.

3.2 Sampled-Data Preliminaries

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*. The reason ϵ is included is that it represents the initial state of the system, which is usually known. 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:

$$L_{conc} = \Sigma^*_{act}.tick \subset L_{samp}$$

We note that if events occur in different sampling periods, we can distinguish which event occurred first. However, if two events occur in the same concurrent string, we can't distinguish order. This means we only consider events in the same concurrent string as occurring concurrently.

Sampled strings represent observable points in the system. If the controller is implementing 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:

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

However, if two concurrent 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 10. For $s \in \Sigma^*$, the *occurrence* operator, Occu : $\Sigma^* \to \text{Pwr}(\Sigma)$, is defined as:

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

To convert TDES $\mathbf{S}$ into SD controller $\mathbf{C}$, we take the sampled states of $\mathbf{S}$ as the states of $\mathbf{C}$. The initial, or *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, i.e. the sampled state in $\mathbf{S}$ that the concurrent string takes us to.

For state x of $\mathbf{C}$, an output (enablement of some $\sigma \in \Sigma_{hib}$) is set to to be true if the corresponding event is possible at x in $\mathbf{S}$. We note that outputs (enablement information) are constant for the clock period. Also, forcing decisions for the clock period are made immediately after the *tick* event occurs. For a formal definition of SD controllers and the conversion process, see [Wang(2009)].

Clearly, our translation method will not work if two concurrent strings with the same occurrence image are possible at a given sampled state, but they lead to different states in $\mathbf{S}$. To ensure this doesn't happen, we require our TDES be *concurrent string deterministic*.

Definition 11. A TDES $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ is *concurrent string (CS) deterministic*, if

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

In Fig. 2, 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. If the two states were not equivalent, we wouldn't be able to translate it.

As stated in our assumptions, controllers only allow prohibitable events to occur once per sampling period, so we want our TDES plant $\mathbf{G}$ to reflect this.

Definition 12. For TDES $\mathbf{G}$ and TDES $\mathbf{S}$, we say that $\mathbf{G}$ has $\mathbf{S}$-*singular prohibitable behavior* if

$$\begin{aligned}&(\forall s \in L(\mathbf{S}) \cap L(\mathbf{G}) \cap L_{samp})(\forall s' \in \Sigma^*_{act}) ss' \in L(\mathbf{S}) \cap L(\mathbf{G})\\ &\quad \implies (\forall \sigma \in \text{Occu}(s') \cap \Sigma_{hib})\ \sigma \notin \text{Elig}_{L(\mathbf{G})}(ss')\end{aligned}$$

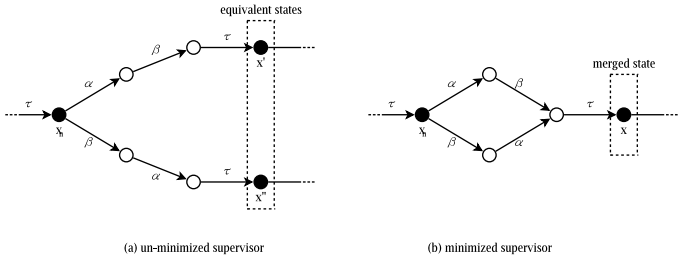

Fig. 2. CS Deterministic Example

3.3 SD Controllable Languages

So far, we have required that our TDES system have a finite statespace, be ALF and nonblocking, that our plant have proper time behavior and be complete for our supervisor, and that our supervisor be controllable for our plant. However, these conditions are not sufficient to address the concerns that we raised in Section 1.1. In particular, even though the above conditions are met, our actual system behavior under the control of the corresponding SD controller could block, violate our control law, or even exhibit behavior not contained in our plant model.

To address these issues, we now introduce a new concept called *SD controllable*. Let $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$ be our plant and $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ be our supervisor. Note that implicit in the definition is the assumption $\Sigma_{for} = \Sigma_{hib}$.

Definition 13. TDES $\mathbf{S}$ is *SD controllable* with respect to TDES $\mathbf{G}$ if, $\forall s \in L(\mathbf{S}) \cap L(\mathbf{G})$, the following statements are satisfied:

(i) $\text{Elig}_{L(\mathbf{G})}(s) \cap \Sigma_u \subseteq \text{Elig}_{L(\mathbf{S})}(s)$

(ii) If $\tau \in \text{Elig}_{L(\mathbf{G})}(s)$ then
$\tau \in \text{Elig}_{L(\mathbf{S})}(s) \Leftrightarrow \text{Elig}_{L(\mathbf{S}) \cap L(\mathbf{G})}(s) \cap \Sigma_{hib} = \emptyset$

(iii) If $s \in L_{samp}$ then
(1) $(\forall s' \in \Sigma^*_{act})[ss' \in L(\mathbf{S}) \cap L(\mathbf{G})] \implies$
$[\text{Elig}_{L(\mathbf{S}) \cap L(\mathbf{G})}(ss') \cup \text{Occu}(s')] \cap \Sigma_{hib} =$
$\text{Elig}_{L(\mathbf{S}) \cap L(\mathbf{G})}(s) \cap \Sigma_{hib}$
(2) $(\forall s', s'' \in L_{conc})$
$[ss', ss'' \in L(\mathbf{S}) \cap L(\mathbf{G}) \wedge \text{Occu}(s') = \text{Occu}(s'')] \Rightarrow$
$ss' \equiv_{L(\mathbf{S}) \cap L(\mathbf{G})} ss'' \wedge ss' \equiv_{L_m(\mathbf{S}) \cap L_m(\mathbf{G})} ss''$

(iv) $L_m(\mathbf{S}) \cap L_m(\mathbf{G}) \subseteq L_{samp}$

Point i: This is standard untimed controllability.

Point ii: The ($\Leftarrow$) direction and **Point i** imply standard TDES controllability. The ($\Rightarrow$) part says that if we enable a prohibitable event, we must disable *tick*. This is to capture the notion that we only enable a prohibitable event when we want to force it. This makes converting TDES $\mathbf{S}$ into an SD controller less ambiguous.

Fig. 3 shows an example satisfying **Point ii**. Our supervisor enables α after first *tick*, then must disable the *tick* there to satisfy ($\Rightarrow$) of **Point ii**. The ($\Leftarrow$) part states we are allowed to disable *tick* here since $\alpha \in \Sigma_{hib}$ is possible to preempt the *tick*.

Point iii: These subpoints apply to the clock period delineated by the sampled string s and the next *tick* event.

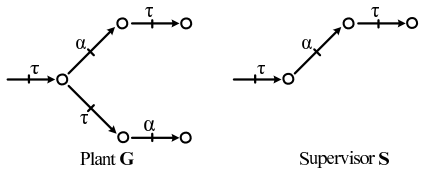

Fig. 3. An Example for **Point ii**

Point iii.1: This point says that when a prohibitable event is possible in a clock period, it must be possible immediately after the *tick* and stay possible for the period until it occurs. This captures the notion that the enablement information for an SD controller is constant for the clock period. It also captures the idea that when a controller forces a prohibitable event, the event must occur before the next *tick*, but we don't know when. This means that the event must be possible in the plant for the entire clock period till it occurs, and must be able to interleave with the other events occurring in the clock period. Otherwise, we could get behavior not in our plant model.

Fig. 4 shows a system that passes **Point iii.1**. We see that the only prohibitable event possible after the *tick* is β, that β stays possible until it occurs on both paths, and no new prohibitable events become eligible before the next *tick*.

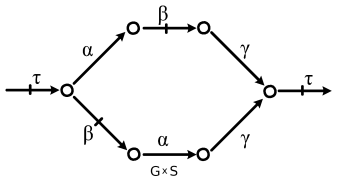

Fig. 4. **Point iii.1**

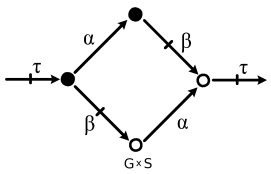

Fig. 5. **Point iv**

Point iii.2: This point says that if two concurrent strings with the same occurrence image can occur, they must have the same future with respect to the system's closed behavior (i.e. we take same control action now and in the future for both strings), and with respect to its marked behavior (i.e. the strings are interchangeable with respect to reaching future marked states). If $\mathbf{G} \times \mathbf{S}$ is minimal, this means that the strings lead to the same state.

Point iv This point says marked strings must be sampled strings (ϵ or end in *tick*). Combined with **Point iii.2**, this ensures that if $\mathbf{G} \times \mathbf{S}$ is nonblocking, then our plant and SD controller will be nonblocking even if only a single concurrent string is possible in our physical system despite our TDES model saying multiple with same occurrence image are possible.

The system in Fig. 5, fails **Point iv**. The left marked state is fine, but the top marked state fails as it is reached by $\tau\alpha$ which is not a sampled string. The concurrent strings $\alpha\beta\tau$ and $\beta\alpha\tau$ have same occurrence image and they take us to the same state so they satisfy **Point iii.2**. Consider the case that in the physical system, we will only get string $\beta\alpha\tau$ and never $\alpha\beta\tau$. If our only marked state was the top marked state, our physical system would never reach it although our TDES model says we could.

4. CONTROLLABILITY AND NONBLOCKING

In this section, we examine how the control action of an SD controller compares to that of the TDES controller it was converted from. In particular, we want to take into account that an SD controller can only update its enablement and forcing information immediately after a *tick* event, while a TDES can be more flexible.

Let $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$ be our plant, and $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ our supervisor. For the rest of this section, we will require plant $\mathbf{G}$ to be complete for $\mathbf{S}$, have proper time behavior and $\mathbf{S}$-singular prohibitable behavior, and that $\mathbf{G} \times \mathbf{S}$ be ALF. Also, that $\mathbf{S}$ be CS deterministic and SD controllable with respect to our $\mathbf{G}$. Let $\mathbf{C}$ be a SD controller translated from $\mathbf{S}$.

4.1 SD Controller as a Supervisory Control

To investigate the enablement and forcing behavior of our controller, we capture this behavior as a supervisory control for $\mathbf{G}$. In [Wang(2009)], we give a detailed algorithm to define map V based on our controller. Here we only have space to give the basic idea. For each $s \in L(\mathbf{G})$, we set $V(s) = \Sigma_u \cup \{\tau\}$. Then for each sampled string s accepted by $\mathbf{G}$ and $\mathbf{C}$ and taking us to state q in $\mathbf{C}$, we add to $V(s)$ the prohibitable events whose corresponding outputs are *true* at state q. We also remove *tick* unless all outputs are *false*. Then for each concurrent string t accepted after s by $\mathbf{G}$ and $\mathbf{C}$, we process each strict prefix $t' < t$. We add to $V(st')$ the same prohibitable events as for s, and remove *tick* if any of these events have not occurred in t'.

4.2 SD Controllers and Controllability

We will now show that the closed-loop behavior $L(V/\mathbf{G})$ equals the behavior of $\mathbf{G} \times \mathbf{S}$, if the TDES satisfy the indicated conditions. This means that we can implement our TDES supervisor as an SD controller and get our expected behavior despite the discussed limitations of the SD controller, at least with respect to the required enablement and forcing actions of the controller.

Theorem 4.1. For plant $\mathbf{G} = (Y, \Sigma, \delta, y_o, Y_m)$, and CS deterministic supervisor $\mathbf{S} = (X, \Sigma, \delta, x_o, X_m)$ that is SD controllable for $\mathbf{G}$, let both TDES have finite statespaces, let $\mathbf{G}$ be complete for $\mathbf{S}$, have proper time and $\mathbf{S}$-singular prohibitable behavior, let $\mathbf{G} \times \mathbf{S}$ be ALF, let $\mathbf{C}$ be the SD controller that is constructed from $\mathbf{S}$, and let V be the map that is constructed from $\mathbf{C}$. Then,

$$L(V/\mathbf{G}) = L(\mathbf{S}) \cap L(\mathbf{G})$$

Proof. See proof in [Wang(2009)]. □

4.3 SD Controllers and Nonblocking

For plant $\mathbf{G}$, and CS deterministic supervisor $\mathbf{S}$ that is SD controllable for $\mathbf{G}$, let $\mathbf{C}$ be the SD controller that is constructed from $\mathbf{S}$, and V be the map that is constructed from $\mathbf{C}$. The *marked behavior* of $V/\mathbf{G}$ is defined to be

$$L_m(V/\mathbf{G}) := L(V/\mathbf{G}) \cap L_m(\mathbf{S}) \cap L_m(\mathbf{G})$$

We say V is *nonblocking* for $\mathbf{G}$ if $\overline{L_m(V/\mathbf{G})} = L(V/\mathbf{G})$. It follows immediately that if the assumptions of Theorem

4.1 are met, then $L_m(V/\mathbf{G}) = L_m(\mathbf{S}) \cap L_m(\mathbf{G})$; thus V is *nonblocking* for $\mathbf{G}$ if and only if $\mathbf{G} \times \mathbf{S}$ is nonblocking.

We also want to ensure that if $\mathbf{G} \times \mathbf{S}$ is nonblocking, then our plant and SD controller will be nonblocking even if only a single concurrent string is actually possible in our physical system at a given sampled state, despite our TDES model saying multiple concurrent strings with the same occurrence image are possible. We wish to be robust with respect to such variations and nonblocking. We will frame our argument in terms of supervisory control maps.

Definition 14. Let $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$ be a TDES and let V and V' be supervisory controls for $\mathbf{G}$. We say V' is *concurrent supervisory control equivalent (CSCE)* to V if

(1) $(\forall s \in L(\mathbf{G}))V'(s) \subseteq V(s)$
(2) $(\forall s \in L(V'/\mathbf{G}) \cap L_{samp})(\forall s' \in L_{conc})ss' \in L(V/\mathbf{G})$
$\implies (\exists s'' \in L_{conc})ss'' \in L(V'/\mathbf{G}) \wedge \text{Occu}(s') = \text{Occu}(s'')$

The first point essentially guarantees that $L(V'/\mathbf{G}) \subseteq L(V/\mathbf{G})$. The second point says that if $V'/\mathbf{G}$ accepts sampled string s, and then $V/\mathbf{G}$ accepts concurrent string s', then $V'/\mathbf{G}$ must accept a concurrent string s'' that has the same occurrence image as s'. Fig. 6 shows a CSCE example. Here, $V/\mathbf{G}$ has two paths with the same occurrence image but $V'/\mathbf{G}$ only one, but that is sufficient.

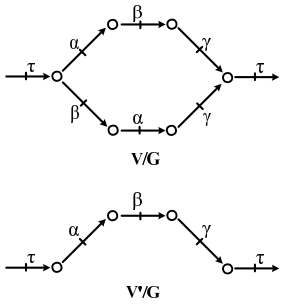

Fig. 6. An Example for CSCE

Theorem 4.2. For plant $\mathbf{G} = (Q, \Sigma, \delta, q_o, Q_m)$, and CS deterministic supervisor $\mathbf{S} = (X, \Sigma, \xi, x_o, X_m)$ that is SD controllable for $\mathbf{G}$, let both TDES have finite statespaces, let $\mathbf{G}$ be complete for $\mathbf{S}$, and have proper time and $\mathbf{S}$-singular prohibitable behavior, let $\mathbf{G} \times \mathbf{S}$ be ALF, let $\mathbf{C}$ be the SD controller that is constructed from $\mathbf{S}$, let V be the map constructed from $\mathbf{C}$ and let V' be a supervisory control for $\mathbf{G}$. If V is nonblocking for $\mathbf{G}$ and V' is CSCE to V, then V' is also nonblocking for $\mathbf{G}$.

Proof. See proof in [Wang(2009)]. □

5. FMS EXAMPLE

In this section we present an example based on the untimed Flexible Manufacturing System (FMS) from [Hill(2008)]. The system, shown in Fig. 7, 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. Table 1 below shows an explanation of event labels used.

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**). The flow of material is illustrated in Fig. 7. Event *921* represents parts entering system, and events *966* and *964* represent finished parts leaving the system. In total, we have 10 plant TDES and 15 modular supervisor TDES. Unfortunately, there is no space to discuss them in detail. See [Wang(2009)] for full details of all TDES, and their design.

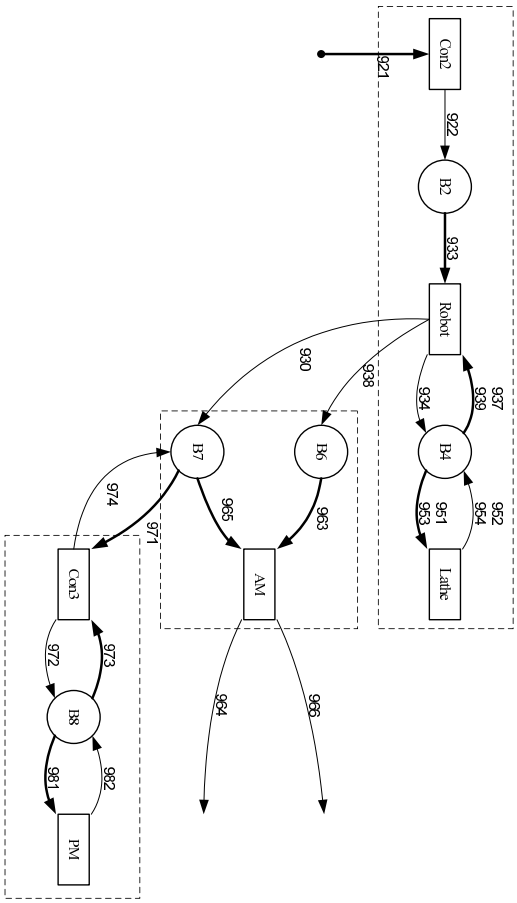

Fig. 7. Flexible Manufacturing System Overview

Table 1. 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		

As part of this work, we developed a set of predicate-based algorithms to verify the SD controllability property, as well as the other conditions that we require. See [Wang(2009)] for details. We have also created a software tool that implements these algorithms using binary decision diagrams (BDD) [Bryant(1992)].

Using a 1.8GHz PC, we verified in 3 minutes that our supervisor **S** is SD controllable for our plant, our plant has proper time behavior, **S**-singular prohibitable behavior, and is complete for our supervisor, and that our closed-loop system (82,608 states) is ALF and nonblocking.

6. CONCLUSIONS

This paper focuses on concurrency and timing issues related to implementing timed discrete-event systems (TDES) as sampled-data (SD) controllers. We first determined existing TDES properties that our system needed to satisfy.

To these existing conditions we added the new requirement that our plant have **S**-singular prohibitable behavior. We then extended TDES controllability to SD controllability, which captures several new properties that are useful in dealing with concurrency issues, as well as makes it easier to translate a supervisor into an SD controller.

Using these properties, we were able to show that the closed-loop behavior of the SD controller and the plant is the same as that of the plant and the original TDES supervisor, at least as far as enablement and forcing goes. We also showed that our method is robust with respect to nonblocking and certain variations in the actual behavior of our physical system.

We applied our approach to a small, flexible, manufacturing system from the literature. We used a software tool we developed to verify that our example passed our conditions, showing that they can be applied in practice.

7. FUTURE WORK

In this paper, we only partly deal with timing delay issues which we have left as future work due to time constraints. We have tried to mitigate potential problems by the assumptions given in Section 3.1. Here, we have required that our controllers be implemented on a single machine, so they have a common clock and their inputs and outputs change at the same time. These address timing issues caused by a distributed implementation.

Another potential timing delay problem is the difference between when an event physically occurs, and when a controller sees that the event has occurred. We have tried to compensate for this by our fourth assumption.

Whereas the steps we have taken to compensate for timing delay are not ideal, they should handle the more pressing issues. However, research needs to be done to address such issues directly in a more flexible manner.

REFERENCES

[Balemi(1994)] Balemi, S. (1994). Input/output discrete event processes and communication delays. *Discrete Event Dynamic Systems*, 4(1), 41–85.

[Basile and Chiacchio(2007)] Basile, F. and Chiacchio, P. (2007). On the implementation of supervised control of discrete event systems. *Control Systems Technology, IEEE Transactions on*, 15(4), 725–739.

[Bolton(2006)] Bolton, W. (2006). *Programmable Logic Controllers*. Elsevier, 4th edition.

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

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

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

[Dragert et al.(2008)Dragert, Dingel, and Rudie] Dragert, C., Dingel, J., and Rudie, K. (2008). Generation of concurrency control code using discrete-event systems theory. In *Proc. of the 16th ACM SIGSOFT Int. Symp. on Foundations of Sftw Eng.*, 146–157. ACM, Atlanta, Georgia.

[Giua and Seatzu(2008)] Giua, A. and Seatzu, C. (2008). Modeling and supervisory control of railway networks using petri nets. *IEEE Transactions on Automation Science and Engineering*, 5(3), 431–445.

[Hill(2008)] Hill, R.C. (2008). *Modular Verification and Supervisory Controller Design for Discrete-Event Systems Using Abstraction and Incremental Construction*. Ph.D. thesis, Department of Mechanical Engineering, University of Michigan.

[Leduc(1996)] Leduc, R. (1996). *PLC Implementation of a DES Supervisor for a Manufacturing Testbed: An Implementation Perspective*. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ont.

[Malik(2003)] Malik, P. (2003). *From Supervisory Control to Nonblocking Controllers for Discrete Event Systems*. Ph.D. thesis, Dept. of Computer Science, University of Kaiserslautern, Kaiserslautern.

[Nourelfath and Niel(2004)] Nourelfath, M. and Niel, E. (2004). Modular supervisory control of an experimental automated manufacturing system. *Control Engineering Practice*, 12(2), 205–216.

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

[Schmidt and Schmidt(2008)] Schmidt, K. and Schmidt, E. (2008). Communication of distributed discrete-event supervisors on a switched network. *Proc. of WODES 2008*, 419–424.

[Wang(2009)] Wang, Y. (2009). *Sampled-data Supervisory Control*. Master's thesis, Dept. of Computing and Software, McMaster University, Hamilton, Ont. [ONLINE] http://www.cas.mcmaster.ca/~leduc/.

[Wong and Wonham(1996)] Wong, K.C. and Wonham, W.M. (1996). Hierarchical control of timed discrete-event systems. *Discrete Event Dynamic Systems*, 6(3), Pages 275 – 306.

[Wonham(2008)] Wonham, W.M. (2008). *Supervisory Control of Discrete-Event Systems*. Department of Elec and Comp Eng, University of Toronto.

[Wonham and Ramadge(1987)] Wonham, W.M. and Ramadge, P. (1987). On the supremal controllable sublanguage of a given language. *SIAM J. Control Optim*, 25(3), 637–659.

[Xu and Kumar(2008)] Xu, S. and Kumar, R. (2008). Asynchronous implementation of synchronous discrete event control. *Proc. of WODES 2008*, 181–186.