Stable supervisory control of flexible manufacturing systems with fixed supply and demand rates

PETER C. Y. CHEN* and W. M. WONHAM†

In this article, we develop a method for computing a stable supervisory control for a class of manufacturing systems (consisting of reconfigurable processors and buffers with limited capacities) with fixed supply and demand rates. It is shown that, with this method, we are able to (i) decide constructively whether a stable supervisory control exists for a given manufacturing system, and (ii) compute the stable supervisory control if it exists. Furthermore, the stable supervisory control thus obtained is complete in the sense that it contains all "safe" sequences (of possibly concurrent buffer and processor activities) which guarantee the stability of the system. An example is presented to illustrate the proposed method. Implications and computational complexity associated with this method are discussed.

1. Introduction

We consider the problem of controlling flexible manufacturing systems consisting of reconfigurable processors and buffers with limited capacities. We describe a manufacturing system as flexible if its processors can be reconfigured to perform different types of tasks. A processor is reconfigurable if its characteristics (physical or otherwise) can be altered in order to perform a given task. For example, a Computer-Numerical-Control (CNC) machine with a set of fixtures can be considered as a reconfigurable system, since performing a new task would involve changing fixtures and activation of a particular CNC program. Reconfiguration of a processor usually requires a certain amount of time.

The dynamics of such manufacturing systems is characterized by the requirement that the rate at which new parts enter the system (i.e., the supply rate) and the rate at which finished parts leave the system (i.e., the demand rate) are fixed. The stability of such a system is characterized by the requirement that the capacities of the buffers must not be exceeded during operation. Since a processor must receive parts from and deliver parts to the buffers, the supply and demand rates thus impose a timing constraint on the operation of the processor. It is in this context that we define a manufacturing system as *stable* if the capacity of the individual buffers within the system is not exceeded during operation.

Management of such systems becomes complicated when a processor is required to perform different tasks on different types of parts. For instance, suppose a processor is required to perform different tasks on two types of parts, each having its own input and output buffers and its own supply and demand rates. To maintain the prescribed supply and demand rates while ensuring that the buffers do not overflow, the processor must be reconfigured so that a certain mix of the two types of parts is processed. The mechanism which manages the activities of the processor is referred to as a supervisory control. By making timely decisions on when to start

---

[*] Bachelor of Technology Programme, Faculty of Engineering, National University of Singapore, EA-07-32, 9 Engineering Drive 1, Singapore 117576. Email: engchenp@nus.edu.sg

[†] Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, Ontario, Canada M5S 3G4. Email: wonham@control.toronto.edu

processing a part and when to reconfigure a processor, a stable supervisory control guarantees that the buffers do not overflow during operation.

The fixed supply and demand rates of a manufacturing system have a significant interpretation under the concept of "hard-real-time manufacturing". Hard-real-time manufacturing differs from conventional approaches to management of manufacturing operations in that time is a critical aspect of operation. To support hard-real-time operation, a manufacturing system must *guarantee* the completion of a given task by a specific deadline. Failure to meet such deadlines is not acceptable. The benefits of adopting the hard-real-time approach to manufacturing include: (i) that manufacturing resources can be more efficiently allocated because the exact time and duration of utilization of a device are known, (ii) that planning of enterprise-wide activities can be carried out more effectively because manufacturing tasks are guaranteed to be completed by specific deadlines, and (iii) that production flow becomes more manageable because problems associated with tardiness of task execution can be avoided.

A manufacturing system operating at a supply rate of one part per t_1 minutes and a demand rate of one part per t_2 minutes is one example of hard-real-time manufacturing. The system must take in one new part every t_1 minutes, and must deliver a finished part every t_2 minutes. The key point is that the intake or the delivery of a part must occur between closely spaced time instants; neither earlier nor later is acceptable.

In this article, we develop a method for computing a stable supervisory control for manufacturing systems with fixed supply and demand rates. Our method is based on the theory of supervisory control of discrete-event systems (DES). A DES is a dynamical process which evolves through abrupt occurrence of events. In a logical DES, the system is characterized by a set of states and transitions (also called events) among these states. The behavior of the system is thus described by a sequence of events. The occurrence of an event (caused by some unmodelled mechanism internal or external to the system) instantaneously moves the system to a new state.

In a timed discrete-event system (TDES), the behavior of the system is also influenced by the temporal characteristics of the event sequence. Specifically, in a TDES, the occurrence of an event could be triggered either by some unmodelled mechanism acting on (or in) the system, or by the passage of time. Supervisory control of a TDES in essence means timely disablement of certain events in the transition structure of the TDES such that its behavior meets certain specifications. A supervisor which prevents events from occurring only when absolutely necessary is described as minimally restrictive. The minimally restrictive behavior of a supervisor is represented by the so-called supremal controllable sublanguage.

To investigate the problem of controlling a manufacturing system for hard-real-time operation, we first model the operation of the buffers and the processor as TDES. We then compute the supremal controllable sublanguage for the TDES with respect to the desired timing behavior of the manufacturing system. To determine whether a system can be controlled such that the supply and demand rates are satisfied while ensuring that the buffers do not overflow, we simply check whether the supremal controllable sublanguage corresponding to this TDES is non-empty. If it is non-empty, then it represents all "safe" sequences of buffer and processor operations (possibly concurrent) which guarantee that the manufacturing system will remain stable while maintaining the desired supply and demand rates. In the context of this article, a safe sequence is a sequence that never takes the system to a state from which, after the further execution of events whose possible occurrence cannot be suppressed, the system would deviate from the desired behavior.

The contribution of this work lies in the development of a formal constructive method for hard-real-time control of dynamical manufacturing systems. With this method, we are able to (i) decide constructively whether a stable supervisory control exists for a given manufacturing system, and (ii) compute the stable supervisory control if it exists. Furthermore, the stable supervisory control thus computed is complete, in the sense that it contains all safe sequences of part processing and processor reconfiguration that produce the desired dynamical behavior of the manufacturing system.

The remainder of this article is organized as follows. Section 2 briefly surveys the relevant literature, and formulates the problem under consideration. Section 3 introduces the theory of supervisory control of TDES. Section 4 describes the proposed method for modelling flexible manufacturing systems with fixed supply and demand rates in the TDES framework, and for synthesizing stable supervisory control for such systems. Section 5 presents an example to illustrate the proposed method. Section 6 discusses some implications of this work and the computational complexity associated with the proposed method.

2. Background and problem formulation

Control of manufacturing systems has been extensively investigated over the last few decades, with its origin found mainly in *job-shop scheduling* (e.g., Jackson 1955, Lawler 1983). Such investigations mostly focus on developing methods for managing the activities of various resources to achieve “optimal” operation of the manufacturing system, under the premise that certain tolerance (such as missing a deadline by a certain amount of time) is permitted.

The criteria of optimality range from minimum processing time and maximum utilization of resources, to least penalty paid for completing a task before the deadline or for missing the deadline (e.g., Asano and Ohta 1996, Cheng and Janiak 1994, Choi and Hitomi 1994, Karacapilidis and Pappis 1993, Lipske 1996, Luh and Hoitomt 1993, Perkins and Kuma 1989, Weng and Ventura 1996). For instance, Weng and Ventura (1996) investigate the problem of single-machine earliness-tardiness scheduling for a given common due date with tolerances. The objective was to minimize the mean absolute deviation of job completion times with the due date tolerance varying from job to job. In Perkins and Kumar (1989), a class of scheduling policies is presented for manufacturing systems where the processing sequences of parts are fixed and parts are to be produced at specific rates (though deviation within certain tolerance is allowed), with consideration for reconfiguration time.

As manufacturing systems become more complex, various formal techniques have been introduced to model, analyze, and control the behavior of such systems. Such formal techniques include max-plus algebra and petri nets. Application of max-plus algebra to manufacturing systems addresses the problem of synchronization of the activities of the various components in the system (Cohen *et al.* 1985). By modelling the timing behavior of a system as a set of “linear” equations in the framework of max-plus algebra, a complete characterization of the asymptotic behavior of the system can be computed. Petri nets have been widely used for modeling manufacturing systems, with consideration for characteristics such as conflicts, priorities, mutual exclusions, and shared resources, etc. (Desrochers and Al-Jaar 1997).

Various approaches for modeling, analysis, and control of manufacturing systems appear to be conceptually related to that used in scheduling of computational resources. However, while hard-real-time has been a critical aspect of task scheduling in computer operating systems (e.g., Liu and Layland 1973), managing manufacturing operations in a hard-real-time environment remains an open problem.

In this article, we consider a workcell consisting of a processor P served by one or more input buffers F_i and output buffers H_i, as illustrated in Figure 1. P in general represents the equipment capable of performing certain tasks on certain types of parts. Such equipment could be a single CNC machine run by an operator, or an automated robotic workstation. In the context of the work reported in this article, it suffices to treat P as a single abstract entity, which can process only one part at a time, and whose operation is non-preemptable.

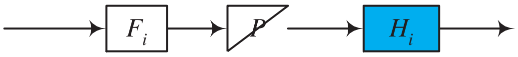


Figure 1. A workcell capable of processing n types of parts.

In the following development, time is assumed to be discrete, and is incremented by a clock tick which is indexed by the natural numbers. Let T be the set of discrete time instants, i.e., $T = \{0, 1, 2, 3, \ldots\}$.

We expect that certain preparation is necessary for P to process a particular type of part. Such preparation may involve configuring the processor in a certain way, such as setting up the correct fixture to hold the parts, etc. Let A be a set of configurations that P can assume, i.e., $A = \{a_\circ, a_1, a_2, \ldots, a_m\}$, where a_i $(i \neq 0)$ is the configuration in which the processor can process type-i parts, and $a_\circ$ is defined as the "neutral" configuration. (The rationale for introducing the neutral configuration is discussed later in Section 4.1.2.) A reconfiguration of the processor from a_i to a_j, $i, j \neq 0$, involves changing the configuration of the processor from a_i to $a_\circ$ and then from $a_\circ$ to a_j. The time required to change a processor from a_i to a_j, referred to as the reconfiguration time and denoted by $r_{ij} \in T$, is defined as follows: $r_{ij} = r_{i\circ} + r_{\circ j} = \Omega(a_i, a_\circ) + \Omega(a_\circ, a_j)$ for $i \neq j$, and $r_{ij} = 0$ for $i = j$, where $\Omega(\cdot) : A \times A \rightarrow T$ is referred to as the *reconfiguration cost function.* Obviously $\Omega(a_i, a_i) = 0$. Informally, $\Omega(\cdot)$ can be considered as a look-up table, which can be constructed by determining the time required for the transition between $a_\circ$ and all the non-neutral configurations.

Let s_i be the rate at which type-i parts enter the workcell. s_i is hence referred to as the *supply rate.* Let d_i be the rate at which type-i parts exit the workcell. d_i is hence referred to as the *demand rate.* For a given part-type i in a period $[0, t]$, let $x_i(t)$ be number of parts that have entered the workcell, $u_i(t)$ the number of parts that have been processed (plus the one being processed, if any) by P, and $y_i(t)$ the number of parts that have exited the workcell. Let $f_i(t)$ and $h_i(t)$ be the buffer levels at F_i (with a maximum capacity of M_i) and H_i (with a maximum capacity of N_i) respectively. Then $x_i(t) = ts_i$, $f_i = x_i(t) - u_i(t)$, and $h_i(t) = u_i(t) - y_i(t)$. As a simple illustration, Figure 2 shows the number of parts fed into the workcell at a rate of one part per two time units, while Figure 3 shows the level of a buffer with a maximum capacity of 4.

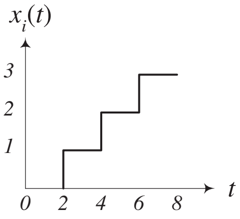


Figure 2. Variation in workcell input.

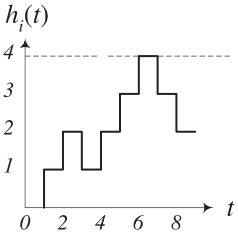


Figure 3. Variation in buffer level.

Suppose there are n types of parts to be processed by P. When the workcell is in operation, type-i parts enter F_i $(i = 1, 2, \ldots, n)$ at a rate of s_i. Since each F_i has a maximum capacity of M_i, to prevent the buffers from exceeding their individual capacity, P is required at some point to start processing type-i parts. This requirement will result in a reconfiguration of P. Such reconfiguration imposes timing constraints on the operation of P, because reconfiguring P from some configuration a_j to the configuration a_i requires r_{ji} units of time, during which parts continue to arrive at and be moved from the buffers.

Supervisory control of the workcell thus entails instructing P when to reconfigure and process a particular type of part. Specifically, the workcell control problem can be stated as follows. Given a set of fixed supply rates s_i, a set of desired fixed demand rates d_i, and a desired throughput D_i for type-i parts, find a supervisory control V such that the following two conditions are satisfied:

1. *On-time delivery of product mix*: The workcell delivers a desired mix of parts at fixed demand rates, i.e.,

$$y_i(t) = \begin{cases} 0 & t \leq t_\circ \\ (t - t_\circ)d_i & t > t_\circ \end{cases} \qquad \text{and} \qquad y_i(\tau) = D_i\,,$$

   where $t_\circ$ is the time instant when the first type-i part is delivered by the output buffer, and $\tau \geq t_\circ > 0$ is the production period.

2. *Internal stability*: The buffers in the workcell do not exceed their individual capacity, i.e., $f_i(t) \leq M_i$ and $h_i(t) \leq N_i$ for $0 \leq t \leq \tau$.

To ensure on-time delivery and stability, the supervisory control must monitor the status and the output of the workcell. This necessitates the use of feedback in the control structure. Figure 4 depicts the structure of the controlled workcell. Our proposed method for synthesizing V is presented in Section 4.

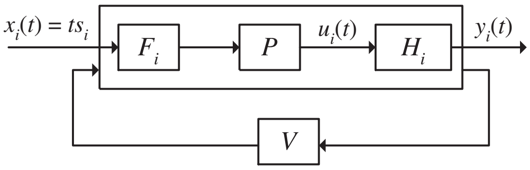


Figure 4. Workcell supervisory control.

Remark: It is noted that the problem under consideration in this work is related to the *Economic Lot Scheduling Problem* (*ELSP*) (Elmaghraby 1978, Lopez and Kingsman 1991), which has been extensively studied in the operational research and production economics literature. The settings of the two problems are similar, but the objectives are different. In our work the objective is to determine a generating mechanism for all safe sequences which meet the given timing constraints, while in the ELSP the objective is to produce a minimum-cost schedule.

3. Supervisory control of timed discrete-event systems

The work reported in this article adopts the framework for supervisory control of timed discrete-event systems originally proposed in Brandin and Wonham (1994). This section summarizes the key concepts essential to the subsequent development. A full exposition of the theory of supervisory control of discrete-event systems is presented in Wonham (1999), which is available on-line at http://www.control.utoronto.ca/people/profs/wonham.

3.1. *Timed discrete-event systems*

A TDES can be expressed as an automaton of the form $G = (Q, \Sigma, \delta, q_\circ, Q_m)$, where Q is a set of states, Σ is a (finite) set of events (including the *tick* event whose occurrence advances a global clock by one unit of time), the partial function $\delta : Q \times \Sigma \to Q$ is a transition function which determines the new state of the system after the occurrence of an event, $q_\circ$ is the initial state of the system, and Q_m is a set of marker states which can be interpreted as the tasks to be completed by the system. Let Σ^* denote the set which contains, in addition to the empty string ϵ, all possible finite sequences over Σ. In the sequel, the transition function δ is extended from the domain $Q \times \Sigma$ to $Q \times \Sigma^*$ according to the following rules: (i) $\delta(q, \epsilon) = q$, and (ii) $\delta(q, u\sigma) = \delta(\delta(q, u), \sigma)$, where $u \in \Sigma^*$ and $\sigma \in \Sigma$, provided that $\delta(q, u)$ and $\delta(\delta(q, u))$ are defined (written as $\delta(q, u)!$ and $\delta(\delta(q, u))!$). A state $q \in Q$ is *reachable* if there is a string $s \in \Sigma^*$ such that $\delta(q_\circ, s) = q$. A state $q \in Q$ is *coreachable* if there is a string $s \in \Sigma^*$ such that $\delta(q, s) \in Q_m$. A TDES is *trim* if all its states are reachable and coreachable.

An event α in a TDES is defined with two time bounds, i.e., $(\alpha,\, l_\alpha,\, u_\alpha)$, where $l_\alpha \in T$ and $u_\alpha \in T$ are the lower and upper time bounds respectively. The interpretation for such a definition is that α may occur (with reference to some point in time) after l_α ticks, and must occur after at most u_α ticks. An event is called *prospective* if $0 \leq l_\alpha \leq u_\alpha < \infty$, and *remote* if $0 \leq l_\alpha < u_\alpha = \infty$.

Each event is thus associated with a timer interval between 0 and the event's uppper time bound, e.g., $T_\sigma = [0, u_\sigma]$ for event σ. A state q of a TDES consists of an activity $b \in B$ (where B is a set of logical states without any reference to time and is referred to as an *activity set*) and a timer $t_\sigma \in T_\sigma$, i.e., $q = (b, \{t_\sigma \,|\, \sigma \in \Sigma_B\})$, where $\Sigma_B = \Sigma - \{tick\}$. The default value for t_σ is u_σ.

The activity set B is equipped with an *activity transition* (partial) *function* $\delta_B : B \times \Sigma_B \rightarrow B$. The TDES transition function $\delta : Q \times \Sigma \rightarrow Q$ for G is defined in terms of δ_B and the time bounds, according to detailed rules given in Brandin and Wonham (1994). An event $\sigma \in \Sigma_B$ is *enabled* at $q = (b, \_) \in Q$ if $\delta_B(b, \sigma)!$, and is disabled otherwise. An enabled event σ (either *tick* or in B) is *eligible* if $\delta(q, \sigma)!$. Only eligible events can actually occur.

The timing mechanism of a TDES can be summarized informally as follows. Once an event α is enabled, the timer t_α of α is decremented by one time unit at every subsequent *tick* of a global clock, until (i) t_α reaches zero (at which point α is forced to occur), or (ii) α occurs, or (iii) α is disabled due to the occurrence of some other transition. In all of these cases, t_α is reset to its default value u_α.

3.2. *Supervisory control*

Supervisory control of a TDES in essence means timely disablement of certain events in the transition structure of the TDES, such that its behavior meets certain specifications. The theories of automata and formal language form the basis for synthesis of supervisory control of TDES.

The language (i.e., the sequences of event labels, or strings) generated by G, denoted by $L(G)$ is called the *closed behavior* of G, i.e., $L(G) = \{s \mid s \in \Sigma^*, \delta(q_\circ, s)!\}$. In other words, $L(G)$ describes all possible transition sequences within the TDES modelled by G. Of all possible sequences, some may be considered as the steps required for completing certain tasks. Such sequences are called the *marked behavior* of G, denoted by $L_m(G)$, i.e., $L_m(G) = \{s \mid s \in \Sigma^*, \delta(q_\circ, s) \in Q_m\}$. By definition, $L_m(G) \subseteq L(G)$.

A string $v \in \Sigma^*$ is called a *prefix* of another string $s \in \Sigma^*$, expressed as $v \leq s$, if $s = vu$ for some $u \in \Sigma^*$. The *prefix closure* of a language $L \subseteq \Sigma^*$, denoted by $\bar{L}$, is defined as $\bar{L} = \{v \in \Sigma^* \mid v \leq s, s \in L\}$. A language is *prefix-closed* if $L = \bar{L}$.

The notion of supervisory control of a TDES is based on the concept of controllability. Controllability is defined in the context that the set of events in a TDES is partitioned into controllable events and uncontrollable events, i.e., $\Sigma_B = \Sigma_c \cup \Sigma_u$, where Σ_c is the set of controllable events and Σ_u is the set of uncontrollable events. An event is controllable if it can be prevented from occurring at a specific point in a logical event sequence or in time, and is uncontrollable otherwise. In a hard real-time system, an event must occur at the latest at the deadline. Supervision of a TDES then must include a mechanism for forcing certain events to occur before a specific time instant. For this reason, an additional type of event, called *forcible* event, is defined to describe events that can preempt a *tick* of the global clock. In this article, we are concerned with tasks having a finite hard deadline. Therefore, all timed events considered in the sequel are forcible unless stated otherwise. More technical definitions of controllable, uncontrollable, and forcible events in the context of TDES can be found in Brandin and Wonham (1994).

The transition structure of a TDES can be visualized as a graph, with the states of the TDES being the nodes, and the events being the edges. At any particular node, there is at least one eligible event, namely *tick*, if no enabled prospective event has timed out. We define a set of (extended) eligible events $H(s)$ with respect to G as $H_G(s) = \{\sigma \in \Sigma \mid s\sigma \in L(G)\}$. Given G over a certain Σ, and a language $K \subseteq L(G)$ with the eligible events $H_K(s) = \{\sigma \in \Sigma \mid s\sigma \in \bar{K}, s \in \Sigma^*\}$, we say that

K is *controllable* with respect to G if, for all $s \in \bar{K}$,

$$H_K(s) \supseteq \begin{cases} H_G(s) \cap \Sigma_u^t & \text{if } H_K(s) \cap \Sigma_f = \emptyset\,, \\ H_G(s) \cap \Sigma_u & \text{if } H_K(s) \cap \Sigma_f \neq \emptyset\,, \end{cases}$$

where $\Sigma_u^t = \Sigma_u \cup \{tick\}$, and Σ_f are the forcible events. This formal definition of controllability can be interpreted intuitively as follows. We can consider a TDES as a generator of a set of "legal" sequences, i.e., sequences that are admitted by the transition structure of the TDES. Thus a TDES is controllable with respect to G if it always admits the occurrence of uncontrollable events eligible in G, and, if *tick* is eligible in G, admits *tick* as well, unless an eligible forcible event is available to preempt it.

A supervisor of a TDES G can be considered as an automaton V that monitors the state of G, and enables or disables certain events in G when necessary so as to influence the behavior of G. Formally, a supervisor is a map $V : L(G) \to 2^{\Sigma}$ such that, for all $s \in L(G)$,

$$V(s) \supseteq \begin{cases} \Sigma_u \cup (\{tick\} \cap H_G(s)) & \text{if } V(s) \cap H_G(s) \cap \Sigma_f = \emptyset\,, \\ \Sigma_u & \text{if } V(s) \cap H_G(s) \cap \Sigma_f \neq \emptyset\,. \end{cases}$$

The closed behavior of G under the supervision of V, denoted by $L(V/G)$, is defined according to (i) $\epsilon \in L(V/G)$, (ii) if $s \in L(V/G)$, $\sigma \in V(s)$, and $s\sigma \in L(G)$, then $s\sigma \in L(V/G)$, and (iii) no other strings belong to $L(V/G)$. The marked behavior of G under V is $L_m(V/G) = L(V/G) \cap L_m(G)$. A supervisor V is *nonblocking* for G if $\bar{L}_m(V/G) = L(V/G)$.

The problem of supervisory control synthesis can be summarized as follows. Given a TDES G over a certain Σ, and some non-empty languages W_1 and W_2 with $W_1 \subseteq W_2 \subseteq L(G)$, find a nonblocking supervisor V such that $W_1 \subseteq L(V/G) \subseteq W_2$. (Conditions for and proof of existence of supervisory control for a given TDES are discussed in detail in Brandin and Wonham (1994). An algorithm for computing V has also been developed in Wonham and Ramadge (1987).) Solution to this problem depends on the controllability of W_2. An important point to emphasize here is that, if W_2 is not controllable, then a largest (i.e., supremal) controllable sublanguage of W_2 can always be found (even though it may be empty). The supervisor V is said to be "minimally restrictive" in the sense that its only action is to disable certain events when necessary so as to preserve the desired behavior of the system. As a consequence, the solution generates the largest possible subset of legal sequences.

A software package called *TTCT* (available from ftp://pc5.isr.uni-stuttgart.de/ttct) has been developed by the Systems Control Group at the University of Toronto. Functionalities of *TTCT* include creating new TDES, combining two or more TDES, and computing the supremal controllable sublanguage of a given language.

4. System modeling and control synthesis

This section describes our proposed method for synthesizing a supervisor for control of a flexible workcell with fixed supply and demand rates. The overall process is illustrated in Figure 5.

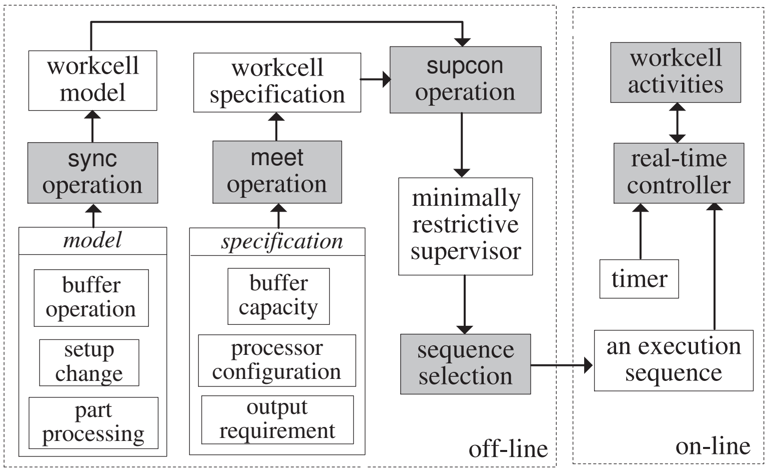


Figure 5. Procedure for computing a supervisor.

The first step is to model the operation of the buffers and the processor as TDES. These TDES can be combined (using the sync operation, i.e., the synchronous product, as discussed below in Section 4.1.3) into a "composite" workcell model. The desired behavior of the workcell is then expressed in a set of specifications. This set consists of three types: one for ensuring that the capacity of a buffer is not exceeded, another for ensuring that the processor is in the proper configuration before processing a part, and the last for ensuring that the workcell output meets certain requirements. These specifications can be combined (using the meet operation as discussed below in Section 4.2.4) into a "composite" workcell specification. Based on the workcell model and the specifications, a minimally restrictive supervisor can be computed (using the supcon operation as discussed below in Section 4.3) which guarantees that the workcell will operate with internal stability while generating the required output.

The result of supcon is the set of all safe sequences of events which can occur without violating the specifications. For on-line implementation, any one of these sequences will suffice to ensure that the workcell operates within specification. Note that for a given workcell model and a set of specifications, it is possible that the resultant supervisor contains no sequence. This simply means that there exists no supervisor which can control the workcell to meet the given specifications.

4.1 *Modeling of workcell operation*

The dynamics of the workcell is represented by the activities of the buffers and the processor. The dynamics of the buffers is characterized by their respective supply or demand rates. The dynamics of the processor consists of two activities: part processing, and reconfiguration. Each of these types of activities can be formally modelled as an automaton.

4.1.1. Buffers

The dynamics of a buffer is described by its throughput rate. Since the number of parts is counted discretely, it is convenient to measure the throughput rate in terms of the time between two consecutive arrivals or departures. That is, a supply rate s or a demand rate d is interpreted to mean 1 part per s or d units of time. s and d are thus implicitly defined as positive integers.

Buffer operation is characterized by two events: one signals the start of part movement, and the other the completion of such movement. Let α denote the first event and β the second event. Depending on whether a buffer is used as an input or output buffer, these events will have different practical interpretation. For an input buffer, the event β has a specific interpretation regardless of implementation details. Its occurrence indicates that a part has physically entered the buffer. The interpretation of the event α, on the other hand, will depend on how part arrival is controlled. Its occurrence, for instance, could mean that the PLC which controls the buffer has sent a request to a main conveyor to dispatch the next part, or that a part has departed from a workcell upstream and is on its way to the current buffer.

For an output buffer, the reverse applies. That is, the event which has a specific interpretation is α (whose occurrence indicates that a processed part has departed from the buffer), while interpretation of the occurrence of β, in this case, will depend on how the movement of the part downstream is controlled.

With suitable interpretation of these events, an input buffer operating at a throughput rate s can be modeled as: $G_F = \{Q_F, \Sigma_F, \delta_F, q_F, X_F\}$, where $Q_F = \{0, 1, 2, \ldots, K\}$, $\Sigma_F = \{t, \alpha_F, \beta_F\}$, $q_F = 0$, $X_F = \{0\}$, and δ_F is defined by: $1 = \delta_F(0, \alpha_F)$, $k = \delta_F((k-1), t)$, and $0 = \delta_F(K, \beta_F)$. k is a positive integer and $1 < k \leq K$, with $K \equiv (s+1)$ numerically. Figure 6 shows the Timed Transition Graph (TTG) of G_F. In a transition graph, the initial state is usually labeled with an entering arrow (i.e., $\longrightarrow\bigcirc$), while a marked state is labeled with an exiting arrow (i.e., $\bigcirc\!\!\longrightarrow$). The double arrow as shown in Figure 6 indicates that state 0 is both an initial and a marked state. The short bar across the transition arrow indicates that the event α_F is controllable. We interpret α_F as controllable in the sense that it may be preempted by other eligible and forcible events, as long as it eventually occurs within its upper time bound. This interpretation of controllable events in the workcell permits the physical realization of the control system (as illustrated in Figure 4), where the supervisor V has sole control over the operations of all workcell devices.

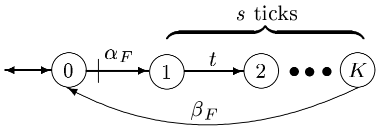


Figure 6. G_F: Input buffer with fixed supply rate.

According to the TDES above, the buffer will immediately start operation at the entrance state. In practice, this means that when the buffer is “switched on” (at the entrance state), an α_F event will occur at once; there can be no delay at that particular state. This behavior is allowed in an input buffer. But for an output buffer, such behavior is not logically correct if initially the output buffer is empty. To resolve this problem, a *tick*-selfloop must be added at the entrance state of the TDES and the transition structure modified slightly, as shown in Figure 7, to ensure that once the very first part has been delivered by the output buffer, the buffer must then operate at a fixed demand rate of d..

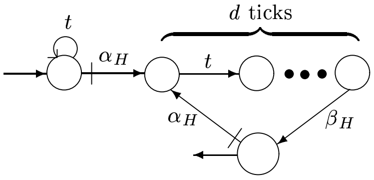


Figure 7. G_H: Output buffer with fixed demand rate.

4.1.2. Processor

The processor can perform two types of tasks: part processing, and reconfiguration. Let γ_i denote the event "processor starts processing a type-i part", and σ_i the event "processor completes processing a type-i part", then the part-processing activity of the processor can be modelled as: $G_{p_i} = \{Q_{p_i}, \Sigma_{p_i}, \delta_{p_i}, q_{p_i}, X_{p_i}\}$, where $Q_{p_i} = \{0, 1, 2, \ldots, K\}$, $\Sigma_{p_i} = \{t, \gamma_i, \sigma_i\}$, $q_{p_i} = 0$, $X_{p_i} = \{0\}$, and δ_{p_i} is defined by: $1 = \delta_{p_i}(0, \gamma_i)$, $k = \delta_{p_i}((k-1), t)$, and $0 = \delta_{p_i}(K, \sigma_i)$. k is a positive integer and $1 < k \leq c_i$, with $K \equiv (c_i + 1)$ numerically and c_i being the time required for processing a part. We normally consider the time for loading the part onto, and for unloading the part from, the processor to be included in c_i. Figure 8 shows the TTG of G_{p_i}.

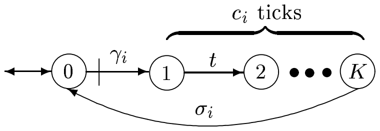


Figure 8. A single task.

A processor may be able to process more than one type of part, but it can process only one part at a time (i.e., its operation is non-preemptable). The TDES model of a processor capable of processing two parts of different types (namely type-i and type-j) is shown in Figure 9.

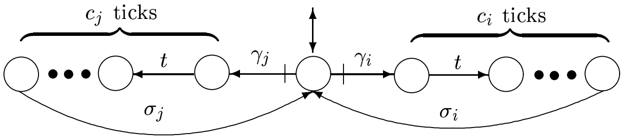


Figure 9. Non-preemptive part processing.

The TDES model of a reconfiguration task is essentially the same as that of a part-processing task: each reconfiguration task requires a certain amount of time, and once started, cannot be interrupted. Let λ_{ij} denote the event "processor starts changing from configuration a_i to configuration a_j", and μ_{ij} the event "processor completes changing from reconfiguration a_i to configuration a_j". Then a TDES G_r of a reconfiguration task can be constructed as shown in Figure 10, where $K \equiv (r_{ij} + 1)$ numerically and r_{ij} is the time required for the reconfiguration from a_i to a_j.

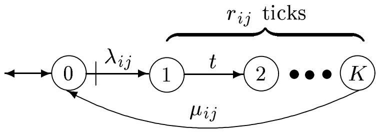


Figure 10. Processor reconfiguration.

If a processor can assume only configurations in which a part can be processed, then there will be a bias towards processing a particular type of part when the workcell is started for the first time. Suppose the initial configuration is a_i; then the built-in bias is to start processing type-i parts, since processing another type of part will immediately incur a reconfiguration cost (in terms of time).

To remove such a bias, we set the initial configuration of the processor as $a_\circ$, from which a reconfiguration must take place before a part can be processed. It is convenient to think of this neutral configuration as a transitional state during a reconfiguration, since changing a processor from configuration a_i to a_j can be interpreted as first from a_i to $a_\circ$ and then from $a_\circ$ to a_j. Figure 11 shows the TDES for setting up a processor between two configurations.

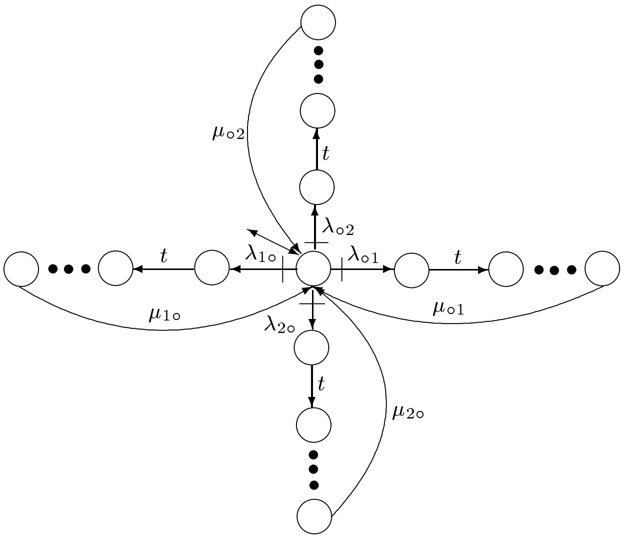


Figure 11. Two configuration tasks.

The introduction of $a_\circ$ does not necessarily prevent the processor from being started in a particular desired configuration. The neutral configuration is needed to remove any inherent bias in the way the workcell may operate, yet within this framework it is still possible for a processor to be started by design in a particular configuration. This can be accomplished by setting $r_{\circ i} = r_{i\circ} = 0$ (i.e., $\Omega(a_\circ, a_i) = \Omega(a_i, a_\circ) = 0$). It should be noted that if $r_{\circ i} \neq 0$ or $r_{i\circ} \neq 0$, then repeating either of $\lambda_{\circ i}\mu_{\circ i}$ or $\lambda_{i\circ}\mu_{i\circ}$ will consume time, which affects the ability of the workcell to meet the timing constraints imposed by the supply and demand rates. With $r_{\circ i} = r_{i\circ} = 0$, the sequences $\lambda_{\circ i}\mu_{\circ i}$ and $\lambda_{i\circ}\mu_{i\circ}$ can occur instantaneously. This may result in the processor being trapped in an instantaneous infinite loop in one or both sequences. However, such loops can be avoided by adding a suitable specification (as discussed in Section 4.2.3 below) to ensure that at least one γ_i will take place after the occurrence of a $\mu_{\circ i}$.

4.1.3. Workcell model

We consider a workcell, as illustrated in Figure 1, capable of processing n types of parts. Suppose that for each type of part, there is one input buffer and one output buffer. Then there are n TDES for the input buffers (namely, G_{F_i}, $i = 1, \ldots, n$) and n TDES for the output buffers (namely, G_{H_i}, $i = 1, \ldots, n$). The TDES for part processing (namely G_p) has n distinct loops, one for each type of part, while the TDES for reconfiguration (namely G_r) has $2n$ distinct loops starting from the neutral configuration.

The initial condition of the workcell is that the processor is idle and all buffers are empty. After start-up, parts begin to arrive (at the rate of one part per s time units) at the input buffers. The processor may subsequently be activated to process the parts, thus triggering the output buffers to deliver parts at the rate of one part per d time units. As a result, there is a time gap between the start of the input buffer and the start of the processor, and subsequently, the start of the output buffer. This time gap has been considered in the TDES model of an output buffer as discussed in Section 4.1.1. For the processor, an additional *tick*-selfloop (i.e., $0 = \delta(0, t)$) is required at the entrance state of G_p and G_r.

A TDES model of the workcell can be constructed by combining G_{F_i}, G_{H_i}, G_p, and G_r into a composite TDES G_w. One way to construct such a TDES is to apply the *synchronous product* (Wonham 1999) over these individual TDES. Given two languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ with $\Sigma = \Sigma_1 \cup \Sigma_2$ and the *natural projection* $P_i : \Sigma^* \to \Sigma_i^*$ defined by: (i) $P_i(\epsilon) = \epsilon$, (ii) $P_i(\sigma) = \epsilon$ if $\sigma \notin \Sigma_i$, (iii) $P_i(\sigma) = \sigma$ if $\sigma \in \Sigma_i$, and (iv) $P_i(s\sigma) = P_i(s)P_i(\sigma)$, $s \in \Sigma^*$ and $\sigma \in \sigma^*$, the synchronous product of L_1 and L_2, denoted by $L_1 || L_2$, is defined as $L_1 || L_2 = P_1^{-1} L_1 \cap P_2^{-1} L_2$. In essence, $L_1 || L_2$ is generated by synchronizing the events that L_1 and L_2 possess in common. In the $TTCT$ software, the synchronous product is computed using the procedure `sync`.

Through the synchronous product, the TDES model of the workcell can be constructed as: $G_w = G_{F_1} || \ldots || G_{F_n} || G_{H_1} || \ldots || G_{H_n} || G_p || G_r$, which contains the "free" behavior of the workcell consisting of the buffers and the reconfigurable processor.

4.2. *Specification*

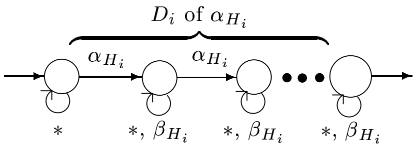
As discussed in Section 2, the workcell must operate to make *on-time delivery* while maintaining *internal stability.* Implicit in this requirement is that the workcell must reconfigure at some point in order to process the parts. These three types of operational specification can be formally expressed in automata.

4.2.1. On-time delivery of product mix

The requirement for on-time delivery can be expressed as

$$y_i(t) = \begin{cases} 0 & t \leq t_\circ \\ (t - t_\circ) d_i & t > t_\circ \end{cases} \qquad \text{and} \qquad y_i(\tau) = D_i \,,$$

where $t_\circ$ is the time instant when the output buffer is activated for the very first time, and $\tau \geq t_\circ > 0$ is the production period. The general TDES (namely S_{O_i}) for this specification is illustrated in Figure 12. The occurrence of one α_{H_i} event signals the departure of one type-i part from the workcell. After D_i consecutive α_{H_i} events the marked state is reached, by which time D_i type-i parts have been delivered.

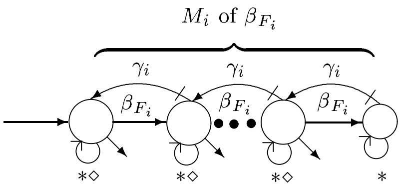


$* =$ all events in Σ_w except α_{H_i} and β_{H_i},
where Σ_w is the event set of G_w

Figure 12. S_{O_i}: Output requirement.

4.2.2. Internal stability

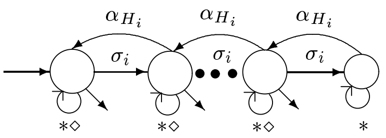
For the workcell to maintain internal stability, none of its buffers must overflow during operation. For an input buffer, its level increases by one after the occurrence of a β_{F_i} event, and decreases by one after the occurrence of a γ_i event (i.e., a part has be removed from the input buffer and loaded onto the processor which subsequently starts processing). For an input buffer with a capacity of M_i not to overflow, there can be at most M_i consecutive occurrences of β_{F_i} before a γ_i must occur. This is expressed formally as a TDES (namely S_{F_i}) as shown in Figure 13.



$\diamond$: α_{F_i}, σ_i
$*$: all events in Σ_w except $\alpha_{F_i}, \beta_{F_i}, \gamma_i$, and σ_i,
where Σ_w is the event set of G_w

Figure 13. S_{F_i}: Capacity of an input buffer.

For an output buffer, its level increases by one after the occurrence of a σ_i event (i.e., a part has be placed on the output buffer after processing), and decreases by one after the occurrence of an α_{H_i} event. For an output buffer with a capacity of N_i not to overflow, there can be at most N_i consecutive occurrences of σ_i before an α_{H_i} must occur. This is expressed formally as a TDES (namely S_{H_i}), as shown in Figure 14.

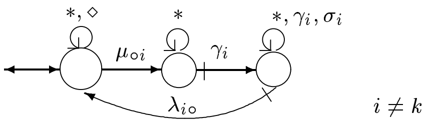


$\diamond$: β_{H_i}, γ_i
$*$: all events in Σ_w except $\alpha_{H_i}, \beta_{H_i}, \gamma_i$, and σ_i,
where Σ_w is the event set of G_w

Figure 14. S_{H_i}: Capacity of an output buffer.

4.2.3. Processor configuration

To process parts of a given type, the processor must be in the corresponding configuration. When it is necessary to process a different type, the processor must reconfigure. To ensure that the processor is in a correct configuration before processing a part, it suffices to require that at least one γ_i event occur between the occurrences of a $\mu_{\circ i}$ event and a $\lambda_{i\circ}$ event. The TDES for this specification (namely S_{P_i}) is illustrated in Figure 15.



$*$: $t, \alpha_{F_i}, \beta_{F_i}, \alpha_{F_k}, \beta_{F_k}, \alpha_{H_i}, \beta_{H_i}, \alpha_{H_k}, \beta_{H_k}$

$\diamond$: $\lambda_{\circ k}, \mu_{k\circ}, \lambda_{\circ i}, \mu_{i\circ}$

Figure 15. S_{P_i}: Proper configuration for part processing.

This specification also prevents the instantaneous infinite loop of reconfiguration (as discussed in Section 4.1.2) from occurring, because between changing from $a_\circ$ to a_i and back to $a_\circ$, the processor must complete at least one type-i part, which takes c_i time units.

4.2.4. Composite specification

For a workcell intended to process n types of parts, there will be n of S_{O_i}, S_{F_i}, S_{H_i}, and S_{P_i} each. Hence there is a total of $4n$ specifications, each of which must be satisfied in order to produce the desired behavior of the workcell. These specifications can be combined (by finding their intersection) into a "composite" specification, which is then imposed on the workcell model G_w (discussed in the next section). One way of computing the intersection is the meet operation, denoted by $\sqcap$.

The meet operation on two TDES G_1 and G_2 yields another TDES $G_3 = G_1 \sqcap G_2$, such that $L_m(G_3) = L_m(G_1) \cap L_m(G_2)$, and G_3 is trim. In fact $\sqcap$ is a special case of $\|$ when $\Sigma_1 = \Sigma_2$, that is, all events are considered shared and synchronization is total. Thus the composite specification for a workcell can be constructed by $S_w = S_{O_1} \sqcap \ldots \sqcap S_{O_n} \sqcap S_{F_1} \sqcap \ldots \sqcap S_{F_n} \sqcap S_{H_1} \sqcap \ldots \sqcap H_{F_n} \sqcap S_{P_1} \sqcap \ldots \sqcap S_{P_n}$.

4.3. *Control synthesis*

Given a TDES G and a set of specifications S, a minimally restrictive supervisor V (i.e., the supremal controllable sublanguage for G with respect to S) can be computed by an iterative algorithm based on a fixpoint characterization of a certain operator Ω (Wonham and Ramadge 1987). It can be shown that V is the largest fixpoint of Ω. The maximum number of iterations needed to compute V is finite, and a worst-case bound can be calculated. In the *TTCT* software, this algorithm is implemented by the procedure supcon.

Given a workcell model G_w and a set of specifications S_w, supcon produces a new TDES (namely, V) which represents the supremal controllable sublanguage for G_w with respect to S_w. In this context, we write $V = \Phi(G_w, S_w)$. In fact, V represents all safe sequences of workcell events which guarantee the stability and performance of the workcell. Note that V may be empty, which implies that there exists no supervisor for G_w with respect to S_w. This could be due to various reasons, including the supply and/or demand rates being too high, or the reconfiguration and part-processing too lengthy. If non-empty, V is called a supervisor of G_w in the sense that under the action of V, the behavior of G_w conforms to the constraints imposed by S_w. In other words, V ensures that the workcell accepts type-i parts at a rate of s_i and delivers D_i parts of type-i at a rate of d_i, while maintaining all buffer levels within capacity.

The overall controlled system is illustrated in Figure 16. Formal description of the feedback coupling of G_w and V can be found in Ramadge and Wonham (1989). For the purpose of this article, it suffices to describe the dynamics as follows. Once the workcell starts operation, both G_w and V run concurrently. Only events which are enabled in both G_w and V at the corresponding states of each can actually occur. Thus one can loosely interpret the transition structure of V as the behavior of G_w which survives after S_w has been imposed.

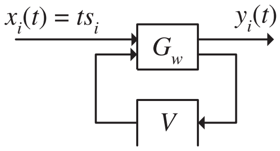


Figure 16. Supervisory control of workcell.

5. An example

We consider a workcell capable of processing two types of parts, designated as type-1 and type-2. The supply and demand rates, the processing time and reconfiguration time are as shown in Figure 17. We impose an output requirement that, for a certain production period (to be determined implicitly by the computation of V), three type-1 parts and two type-2 parts are to be delivered.

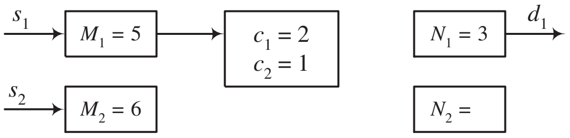


Figure 17. A workcell for processing two types of parts.

Note that with $r_{\circ 1} = r_{1\circ} = 0$ as shown in the table, the processor actually has only two configurations (a_1 and a_2); the neutral configuration $a_\circ$ is "virtual". We take the smallest time resolution to be one minute; thus one *tick* advances the global clock by one minute. The TDES for the input buffers are shown in Figure 18, and those for the output buffers in Figure 19.

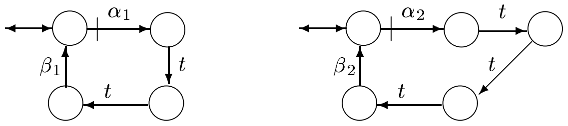


Figure 18. G_{F_1} and G_{F_2}: Input buffers for type-1 and type-2 parts.

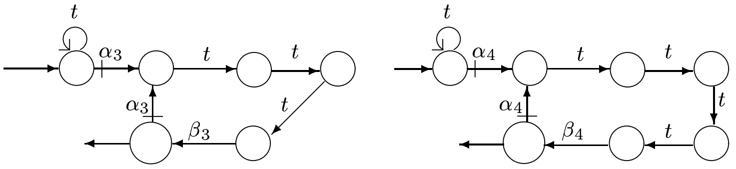


Figure 19. G_{H_1} and G_{H_2}: Output buffers for type-1 and type-2 parts.

The processor is modelled by two TDES, one for part-processing (Figure 20) and the other for reconfiguration (Figure 21). The workcell model G_w is computed as $G_w = G_{F_1} \| G_{F_2} \| G_{H_1} \| G_{H_2} \| G_p \| G_r$. It consists of 35,280 states and 129,792 transitions.

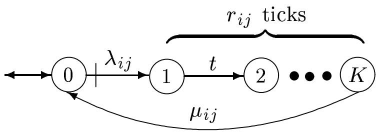


Figure 20. G_p: Part processing.

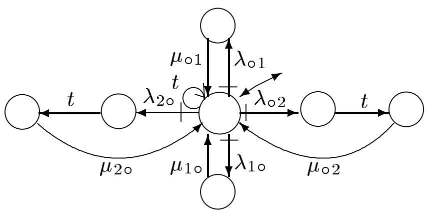
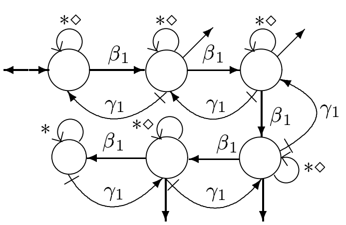


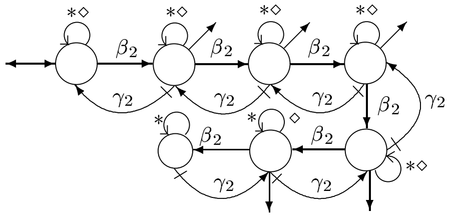
Figure 21. G_r: Processor reconfiguration.

The capacities of the input buffers are $M_1 = 5$ and $M_2 = 6$, and of the output buffers are $N_1 = 3$ and $N_2 = 5$. The respective specifications (namely, S_{F_1}, S_{F_2}, S_{H_1}, and S_{H_2}) for ensuring that the capacities of these buffers are not exceeded are shown in Figures 22 – 25.



$*$: $t, \alpha_2, \beta_2, \alpha_3, \beta_3, \alpha_4, \beta_4, \gamma_2, \sigma_2,$
$\lambda_{\circ 1}, \mu_{\circ 1}, \lambda_{\circ 2}, \mu_{\circ 2}, \lambda_{1\circ}, \mu_{1\circ}, \lambda_{2\circ}, \mu_{2\circ}$
$\diamond$: α_1, σ_1

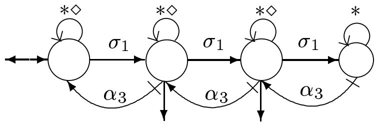
Figure 22. S_{F_1}: Specification for type-1 part input buffer.



$*$: $t, \alpha_1, \beta_1, \alpha_3, \beta_3, \alpha_4, \beta_4, \gamma_1, \sigma_1,$
$\lambda_{\circ 1}, \mu_{\circ 1}, \lambda_{\circ 2}, \mu_{\circ 2}, \lambda_{1\circ}, \mu_{1\circ}, \lambda_{2\circ}, \mu_{2\circ}$

$\diamond$: α_2, σ_2

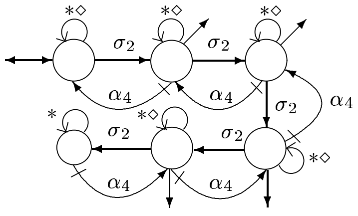
Figure 23. S_{F_2}: Specification for type-2 part input buffer.



$*$: $t, \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_4, \beta_4, \gamma_2, \sigma_2,$
$\lambda_{\circ 1}, \mu_{\circ 1}, \lambda_{\circ 2}, \mu_{\circ 2}, \lambda_{1\circ}, \mu_{1\circ}, \lambda_{2\circ}, \mu_{2\circ}$

$\diamond$: γ_1, β_3

Figure 24. S_{H_1}: Specification for type-1 part output buffer.

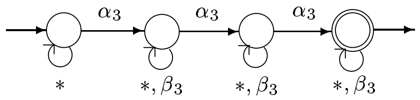


$*$: $t, \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3, \gamma_1, \sigma_1,$
$\lambda_{\circ 1}, \mu_{\circ 1}, \lambda_{\circ 2}, \mu_{\circ 2}, \lambda_{1\circ}, \mu_{1\circ}, \lambda_{2\circ}, \mu_{2\circ}$

$\diamond$: γ_2, β_4

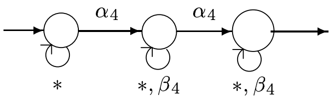
Figure 25. S_{H_2}: Specification for type-2 part output buffer.

The output specifications are shown in Figures 26 and 27. The specification for ensuring that the processor is in the correct configuration before processing a part is shown in Figure 28.



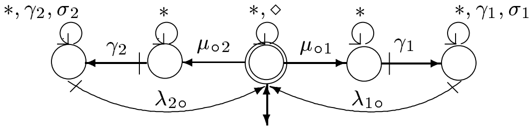
$*$: $t, \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_4, \beta_4, \gamma_1, \sigma_1, \gamma_2, \sigma_2,$
$\lambda_{\circ 1}, \mu_{\circ 1}, \lambda_{\circ 2}, \mu_{\circ 2}, \lambda_{1\circ}, \mu_{1\circ}, \lambda_{2\circ}, \mu_{2\circ}$

Figure 26. S_{O_1}: Output specification for type-1 parts.



$*$: $t, \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3, \gamma_1, \sigma_1, \gamma_2, \sigma_2,$
$\lambda_{\circ 1}, \mu_{\circ 1}, \lambda_{\circ 2}, \mu_{\circ 2}, \lambda_{1\circ}, \mu_{1\circ}, \lambda_{2\circ}, \mu_{2\circ}$

Figure 27. S_{O_2}: Output specification for type-2 parts.



$*$: $t, \alpha_1, \beta_1, \alpha_2, \beta_2, \alpha_3, \beta_3, \alpha_4, \beta_4$
$\diamond$: $\lambda_{\circ 1}, \mu_{1\circ}, \lambda_{\circ 2}, \mu_{2\circ}$

Figure 28. S_P: Proper configuration of processor.

Due to memory limitation (64MB), a supervisor was computed (using the $TTCT$ software) by repeated application of Φ on G_w and the specifications, i.e.,

$$V = \Phi(\Phi(\Phi(\Phi(G_w, S_P), S_{F_1} \sqcap S_{F_2}), S_{H_1} \sqcap S_{H_2}), S_{O_1} \sqcap S_{O_2}).$$

The resulting supervisor V has 2,538 states and 5,945 transitions. It represents all safe sequences of workcell operation which meet all the specifications. As an illustration, one such sequence is traced out as follows: $\alpha_1\alpha_2tt\beta_1\alpha_1t\beta_2\alpha_2t\beta_1\alpha_1\lambda_{\circ 1}\mu_{\circ 1}\gamma_1tt\beta_1$ $\alpha_1\beta_2\alpha_2\sigma_1\gamma_1tt\beta_1\alpha_1\sigma_1$ $\lambda_{1\circ}\mu_{1\circ}\lambda_{\circ 2}t\beta_2$ $\alpha_2\alpha_3\mu_{\circ 2}\gamma_2t\beta_1\alpha_1\sigma_2\alpha_4\gamma_2t\sigma_2\lambda_{2\circ}t\beta_1\alpha_1\beta_2\alpha_2\beta_3\alpha_3\mu_{2\circ}$ $\lambda_{\circ 1}\mu_{\circ 1}\gamma_1tt\beta_1\alpha_1\beta_4\alpha_4\sigma_1\gamma_1t\beta_2\alpha_2\beta_3\alpha_3t\beta_1\alpha_1\sigma_1\gamma_1$ $tt\beta_1\beta_2\beta_3\beta_4\sigma_1\lambda_{1\circ}\mu_{1\circ}$. The buffer levels and the processor activity, when the workcell is run according to this event sequence, are depicted in Figures 29 – 33. By inspection of the sequence and the figures, it can be verified that the operation of the workcell as described by this sequence satisfies all of the specifications.

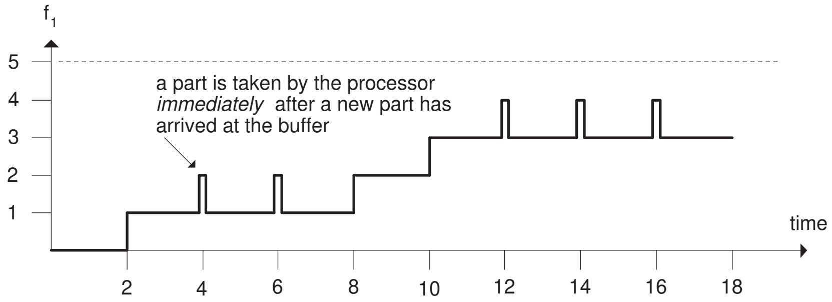


Figure 29. Level of type-1 part input buffer.

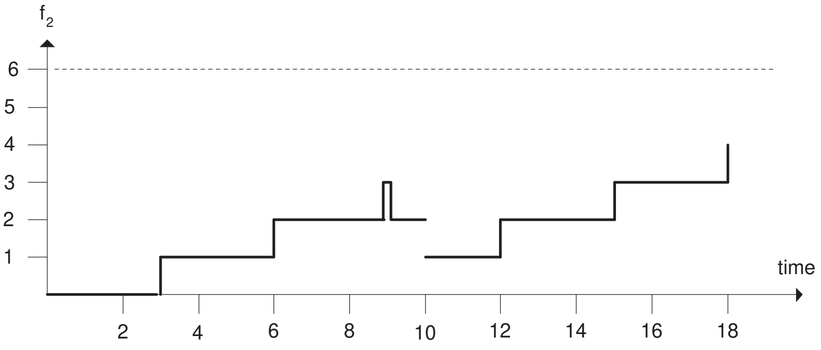


Figure 30. Level of type-2 part input buffer.

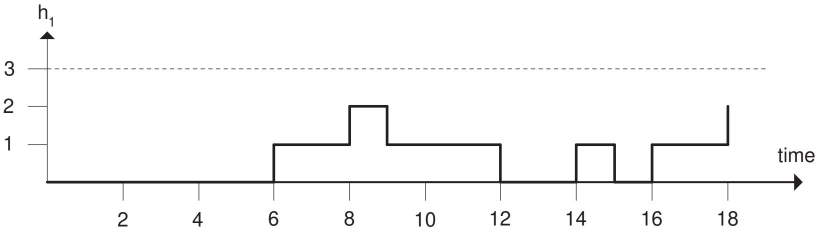


Figure 31. Level of type-1 part output buffer.

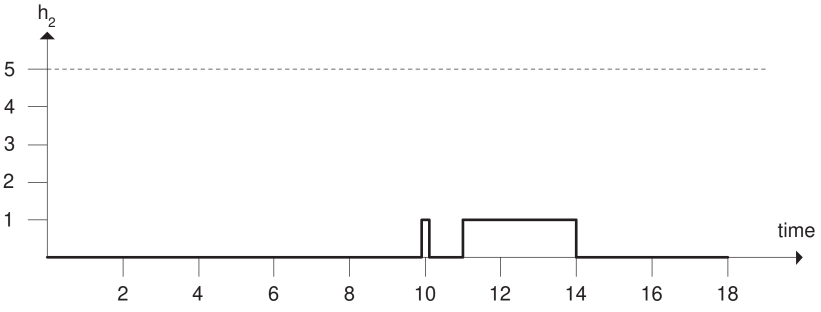


Figure 32. Level of type-2 part output buffer.

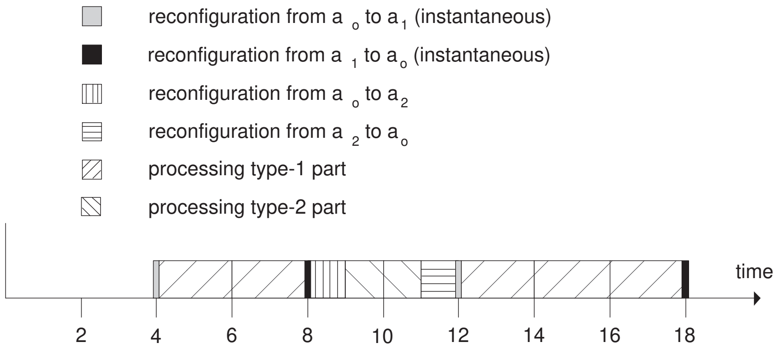


Figure 33. Processor activity under supervisory control.

6. Discussion

We have presented a method for computing a supervisor for a workcell with fixed supply and demand rates. This method is based on the theory of supervisory control of discrete-event systems, which guarantees that the result is complete and (logically and temporally) correct-by-construction. The result is complete in the sense that it not only indicates whether a supervisor exists for a given G_w with S_w, but also contains all safe sequences, if a supervisor exists, for controlling G_w under the constraint of S_w (i.e., the supervisor is minimally restrictive). The fact that the resulting supervisor is minimally restrictive allows the possibility for optimization of the supervisor according to additional criteria, such as the requirement that the workcell should produce a certain mix of parts with a minimum number of reconfigurations.

With such completeness comes computational complexity, which is characterized by (i) the modelling of clock increment as a distinct event in the TDES framework as proposed in Brandin and Wonham (1994), and (ii) the exponential growth in the number of states when combining individual components of the workcell (i.e., the buffers and the processor) into a single TDES using the synchronous product. The TDES framework adopted in this work models time by explicitly adding the *tick* event to represent the increment of the clock by one time unit. If a workcell component has activities which are lengthy compared to the time resolution of the clock, then a significant number of states and transitions will be generated simply due to the passage of time.

It has been shown that the computation of the supremal controllable sublanguage with respect to a finite TDES can be completed in polynomial time (Ramadge and Wonham 1989). The complexity in the synthesis of a supervisory control stems from the fact that, with the synchronous product, the number of states of a composite TDES increases exponentially with the number of components. This remains a challenge in the "scaling up" of the proposed method for computation of the supervisor. Two approaches may be explored to deal with this difficulty. One may be termed a "brute-force" approach, whereby modern relational database technologies and computing systems are employed to deal with the large number of states, transitions, and the associated computation. The other approach, namely modular synthesis (Wonham and Ramadge 1988), seeks to reduce computational complexity by synthesizing a set of modular supervisors which, when operating concurrently, achieve the same result as a centralized supervisor does.

References

Asano, K. and Ohta, H., 1996, Single machine scheduling using dominance relation to minimize earliness subject to ready and due times. *International Journal of Production Economics*, **44**, 35-43.

Brandin, B. A. and Wonham, W. M., 1994, Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, **39**, 329-342.

Cheng, T. C. E. and Janiak, A., 1994, Resource optimal control in some single-machine scheduling problems. *IEEE Transactions on Automatic Control*, **39**, 1243-1246.

Choi, J. and Hitomi, K., 1994, A method of flexible scheduling for flexible manufacturing systems. *International Journal of Production Economics*, **33**, 247-255.

Cohen, G., Dubois, D., Quadrat, J. P., and Viot, M., 1985, A linear-system-theoretical view of discrete-event processes and its use for performance evaluation in manufacturing. *IEEE Transactions on Automatic Control*, **AC-30**, 210-220.

Desrochers, A. A. and Al-Jaar, R. Y., 1997, *Application of Petri Nets in Manufacturing Systems*. (New York: IEEE Press).

Elmaghraby, S., 1978, The economic lot scheduling problem (ELSP): review and extensions. *Management Science*, **24**, 587-598.

Jackson, J. R., 1955, Scheduling a production line to minimize maximum tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles.

Karacapilidis, N. I. and Pappis, C. P., 1993, Optimal due date determination and sequencing of n jobs on a single machine using the SLK method. *Computers in Industry*, **21**, 335-339.

Lawler, E. L., 1983, Recent results in the theory of machine scheduling. In Bachem et. al. (eds), *Mathematical Programming: The State of the Art*, (New York: Springer-Verlag), 202-233.

Lipske, K. R., 1996, A greedy-based decision support system for scheduling a manufacturing operation. *Production and Inventory Management Journal*, **37**, 36-39.

Liu, C. L. and Layland, J. W., 1973, Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, **20**, 46-66.

LOPEZ, M. A. N. and KINGSMAN, B. G., 1991, The economic lot scheduling problem: theory and practice. *International Journal of Production Economics*, **23**, 147-164.

LUH, P. B. and HOITOMT, D. J., 1993, Scheduling of manufacturing systems using the Lagrangian relaxation technique. *IEEE Transactions on Automatic Control*, **38**, 1066-1079.

PERKINS, J. R. and KUMAR, P. R., 1989, Stable, distributed, real-time scheduling of flexible manufacturing/assembly/disassembly systems. *IEEE Transactions on Automatic Control*, **34**, 139-148.

RAMADGE, P. J. and WONHAM, W. M., 1989, The control of discrete event systems. *Proceedings of the IEEE*, *77*, 81-98.

WENG, M. X. and VENTURA, J. A., 1996, Single-machine earliness-tardiness scheduling about a common due date with tolerances. *International Journal of Production Economics*, **42**, 217-227.

WONHAM, W. M. and RAMADGE, P. J., 1987, On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, **25**, 637-659.

WONHAM, W. M. and RAMADGE, P. J., 1988, Modular supervisory control of discrete event systems. *Mathematics of Control, Signal and Systems*, **1**, 13-30.

WONHAM, W. M., 1999, *Notes on Control of Discrete-Event Systems.* Version 99.12.01, Department of Electrical and Computer Engineering, University of Toronto, 1999. (Available on-line at: http:// www.control.utoronto.ca / people / profs / wonham.)