Int J Adv Manuf Technol (2009) 41:1152–1168
DOI 10.1007/s00170-008-1555-9

ORIGINAL ARTICLE

A structural approach to the non-blocking supervisory control of discrete-event systems

Lei Feng · Kai Cai · W. M. Wonham

Received: 27 November 2007 / Accepted: 29 April 2008 / Published online: 3 June 2008

Abstract Many practical and important systemic properties of manufacturing systems, like deadlock freeness, liveness, and reversibility, can be formulated as the non-blocking property of discrete-event systems. It can be difficult, however, to verify non-blocking or design a supervisor to guarantee non-blocking control because of state size explosion in the concurrency model. In this paper, we present sufficient conditions for the computation of (small) model abstractions that preserve the non-blocking property. As a consequence, hierarchical and decentralized control structures can be flexibly integrated, and the proposed approach can synthesize maximally permissive and non-blocking control with reduced computational effort. The solution is a group of decentralized supervisors that transparently displays control logic and admits relatively simple implementation.

Keywords Supervisory control · Discrete-event systems · Non-blocking control

L. Feng (✉)
Embedded Control Systems Research Group,
Department of Machine Design,
KTH-Royal Institute of Technology,
Brinellv. 83,
100 44 Stockholm, Sweden
e-mail: leifeng@md.kth.se

K. Cai (✉) · W. M. Wonham (✉)
Systems Control Group,
Department of Electrical and Computer Engineering,
University of Toronto,
Toronto M5S 3G4 ON, Canada

K. Cai
e-mail: caikai@control.utoronto.ca

W. M. Wonham
e-mail: wonham@control.utoronto.ca

1 Introduction

At a management and logistical level, many industrial systems evolve according to the 'random' occurrence of predefined events rather than, or in addition to, the 'tick' of a clock. From this perspective, several *discrete-event system* (DES) formalisms have emerged [1]. Among them, *supervisory control theory* (SCT), established by Ramadge and Wonham [2, 3] in the 1980s, is a formal framework for the control synthesis of DES. Application domains include manufacturing systems [4, 5], robotics [6], vehicular traffic [7], logistics [8], computer systems [9], and communication networks [10, 11]. Problems that SCT can address include dynamic resource allocation, the coordination of individual control tasks subject to priorities and hard real-time deadlines, the prevention of system blocking and, within these constraints, maximally permissive system behavior. In these applications, SCT can systematically synthesize supervisory controllers that prevent a DES from executing undesirable behavior by disabling certain events in its dynamic structure.

Of practical importance for a DES, like a manufacturing system, is the *non-blocking* property, namely that the system can always reach some target state from any state in its reachable set. Common requirements of manufacturing systems, like *deadlock freeness* and *reversibility*, are instances of the non-blocking property. Owing to computational complexity, these properties are often difficult to guarantee or verify. Generally, all possible states of the system must be examined, while the number of states is exponential in the number of the components included in the system, a phenomenon called *state explosion*. To deal with such complexity, much research effort has been dedicated to the efficient computational design of non-blocking supervisory control [12–18].

While these results can be effective in favorable circumstances, they all point to the conjecture that universal algorithms of complexity *polynomial* in the number of component DES of the plant model are unlikely to be found for non-blocking supervisory control. This view is supported by results showing that the time complexity for the non-blocking control problem is *NP-complete* [19, 20]. In light of this discovery, we need to take the modest approach of seeking tractable methods for various subclasses of DES that enjoy special structure.

Manufacturing systems often display 'product' structure consisting of a group of independent autonomous plant components subject to multiple control specifications. For example, the latter may embody sharing of limited resources, or precedence constraints on operations. Product structure has supported advances in *decentralized* supervisory control [12, 14, 18, 21–23] and *hierarchical model abstraction* [24–27]. In addition, recent investigations [15, 28, 29] have demonstrated that DES with product structure can be efficiently represented by compact *binary decision diagrams* (BDD), allowing synthesis of non-blocking control of systems with state sizes beyond 10^{20}.

Building on these results, we have proposed a unified approach to the computationally efficient supervisor design of DES with product structure [30–34]. Two types of models play a role: graph models, which display the interaction and control flow among plant components and specifications, and automaton models for system dynamics. Graph models are used to simplify the control problem through qualitative reasoning, allowing the automaton models to support explicit control synthesis. We reduce the complexity of automaton computations by means of suitable control architecture and model abstraction. Compared to other methods that also do not use the BDD data structure, our approach (when it succeeds) always achieves optimal and non-blocking control, and generally consumes smaller computational resources. This paper will demonstrate the power of the approach through the detailed control design for a production cell.

Note that the approach is especially designed for DES with product structure, where decomposition and modularity may be easily realized. If a system does not have product structure, either it would be difficult to use the approach, or the reduction in computational complexity might be small. Moreover, because the non-blocking supervisory control problem is NP-complete, our approach, like all others, will eventually suffer the state explosion problem when the system is very large and complex.

While the cited approaches and our own are based on finite state automaton models, there is significant research on control design based on Petri nets (PN) [35, 36]. Possible advantages of PN are compact state representation by integer vectors, and graphical display of the interconnection relationship among system components, supporting qualitative reasoning based on net structure.

On the other hand, non-blocking control design based on PN is generally as complex as the automaton-based approaches; and worse, *optimal* supervisors need not always be representable as PN [37]. A supervisor is said to be *optimal* (or *maximally permissive*) for a given plant with its control specification, if the controlled behavior of the plant is larger (under set inclusion) than that of any other solution for the same pair. Uzam and Wonham [38] have proposed a hybrid approach to optimal and non-blocking automaton supervisors for uncontrolled PN plant models. The two types of models are coupled to complete the feedback loop. Thus, in one way or another, the approach described in the present paper is not restricted to problems modeled only using automata.

Resource allocation graph [39, 40] is another viable approach for the deadlock avoidance of flexible manufacturing systems. Compared to the former two, its limitation is that all plant components are simplified as resources and do not have internal behaviors. Moreover, non-blocking property is stronger than deadlock-freeness. While resource allocation graph is not suitable for our more general supervisory control problem formalized in Sect. 2, they motivate us to analyze system interconnection structure with digraph models [31, 32].

In the sequel, Sect. 2 reviews the preliminaries of SCT. Section 3 introduces a scheme for applying our structural approach to the supervisory control synthesis of product DES, and the theoretical basis of the approach. Sections 4 and 5 describe the production cell example and its DES models. The overall system structure is presented in Sect. 6, while Sects. 7–10 elaborate on the detailed design. Section 11 states the conclusion.

2 Supervisory control theory

As *supervisory control theory* (SCT) is well established, we refer the reader for background to standard textbooks like [41] and [42]. This section reviews only the most basic concepts and notation relevant to this paper. SCT is directly based on *regular* languages and finite state automata. Let Σ be a finite set of events. The empty string of length 0 is denoted ε and the set of all finite strings over Σ including ε denoted Σ^*. For two strings s, t$\in\Sigma^*$, we write $s \leq t$ if s is a *prefix* of t, namely, $t=su$ for some $u\in\Sigma^*$. Given a regular language $L\subseteq\Sigma^*$, the language $\overline{L} := \{s | s \leq t \text{ for some } t \in L\}$ is its *prefix-closure*.

A unique advantage of SCT is the separation of the concept of *plant* (open-loop dynamics) from the *feedback* control so that traditional control theoretic notions such as controllability, observability, modularity, decentralized and hierarchical control, can be exploited. In applications, the

plant is modeled as an automaton (**G**). The event set in **G** is its *alphabet* (Σ). The desirable behavior of the controlled system is determined by a *control specification*, also modeled as an automaton (**E**). Both the plant **G** and the control specification **E** may be the *synchronous product* (concurrent composition) of many smaller modular components.

A group of supervisory controllers (supervisors) closes the loop of a controlled DES and forces the plant to respect the control specifications. The controllers act only to *disable* certain events that are originally able to occur in the plant, thus preventing them from occurring. The control logic of a supervisory controller is derived from the event disablement list at each state. In practice, we may assume that some events in the alphabet can never, or need not be disabled. Such events are called *uncontrollable*, while those preventable by a supervisory controller are called *controllable*. Hence, the alphabet Σ is partitioned into two disjoint subsets, controllable events (Σ_c) and uncontrollable events (Σ_{uc}), such that $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$.

To synthesize a satisfactory supervisor, SCT provides a formal method for theoretically tackling the typical supervisory control problem:

Given a plant **G** *over alphabet* Σ *with its partition* $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$ *and control specifications modeled as* **E**, *find a maximally permissive supervisor* **S** *such that the controlled system* **S**/**G** *is non-blocking and always meets the control specifications.*

To formalize *partial observation*, we partition Σ as observable event set Σ_o and unobservable event set Σ_{uo}, namely, $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. We define the natural projection

$$P : \Sigma^* \to \Sigma_o^*$$

according to

$$P(\sigma) = \begin{cases} \sigma, & \sigma \in \Sigma_o \\ \varepsilon, & \sigma \notin \Sigma_o \end{cases}$$

$P(\varepsilon) = \varepsilon$, and $P(s\sigma)=P(s)P(\sigma)$ for all $s\in\Sigma^*$ and $\sigma\in\Sigma$. The effect of P on a string $s\in\Sigma^*$ is just to erase the events in s that are unobservable and retain the observable ones in the previous order.

3 Application scheme of the new approach

This section briefly outlines our approach to design, starting from a given DES control problem with product structure.

Step 1. Obtain decentralized supervisors for individual specifications. Because a specification often involves only a small number of plant components, the decentralized supervisor for the specification is synthesized based on its 'local' plant [12], namely the synchronous product of just those plant components that share joint events with the specification. Such a supervisor may often be found by inspection.

Step 2. Partition the plant components and decentralized supervisors into modular subsystems according to their interaction dependencies. *Control-flow net* (CFN) [31, 33] and *process communication graph* (PCG) [28] are two effective tools for capturing the interaction dependencies and selecting subsystems appropriately. If each subsystem contains only a few plant components and decentralized supervisors, or admits simple control logic, we can find its optimal and non-blocking control by qualitative reasoning assisted by only modest computation [32]. We have formalized some effective heuristics for this step, and they will be implemented in software in the future. Presently, this step is performed by designer's inspection.

For illustration, consider the system in Fig. 1, which includes four plant components $G_i (i = 1, \cdots, 4)$, and several specifications not displayed. Suppose Step 2 organizes the system into three subsystems $S_i (i=1, 2, 3)$, enclosed by three dashed ovals as shown.

Step 3. Ensure the non-blocking property within each subsystem. After Step 2, each subsystem should contain only a small number of components, or can be described by a simple control-flow net. We can hence find its optimal and non-blocking supervisor with only modest computation. At this stage, if we already know at Step 2 that these subsystems are mutually nonconflicting, then the supervisor design process terminates; otherwise, continue to Step 4.

Step 4. Design the model abstraction of each subsystem using *natural observers* [43, 44]. Abstraction introduces hierarchy into system structure, as it reports only the events shared with other subsystems and conceals the rest. The fewer the reported events, the greater state reduction will be

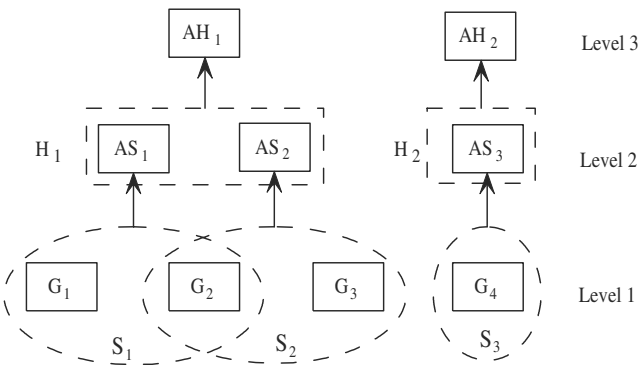


Fig. 1 Control architecture

achieved. The abstractions of the three subsystems in Fig. 1 are $AS_i(i=1, 2, 3)$ displayed at Level 2.

Step 5. Organize these model abstractions into groups according to their interconnections. In simple cases, all the model abstractions will belong to just one group. For each group, check whether the included model abstractions are nonconflicting and, if not, design a coordinator to resolve the conflict. The basis for this step (see [33, 34]), which is the main theoretical contribution of the proposed approach, will only be reviewed informally in the present paper.

In Fig. 1, we organize abstractions AS_1 and AS_2 into group H_1, and AS_3 alone into group H_2. The two groups at Level 2 are marked by dashed squares. Under this setup, we need only study the interaction between AS_1 and AS_2 in H_1.

Step 6. Repeat Steps 4 and 5 by regarding the model abstractions as higher-level plant components and the groups of model abstractions as the subsystems, until there is only one group in Step 5. The result is a hierarchy of decentralized supervisors and coordinators. In Fig. 1, since there are two groups of model abstractions at Level 2, we continue by finding model abstractions for H_1 and H_2. This produces AH_1 and AH_2 at Level 3. Evidently we can place the two abstractions into one group and complete the control design.

It is possible to devise a similar supervisor design method using bottom-up hierarchical model abstraction, but then the challenge is to guarantee that the decentralized supervisors and coordinators so obtained are indeed optimal and non-blocking. The key to this objective is the proper selection of observable events at Step 4. To this end, we have found sufficient conditions [30, 33] on the observable event set of each controlled subsystem. They are the main contribution of our approach. To the best of our knowledge, these conditions are arguably the first computationally efficient ones that systematically guarantee both optimal and non-blocking properties of supervisory control of DES.

Suppose there are n non-blocking subsystems at Step 4, each having alphabet $\Sigma_i(i=1,\cdots,n)$. Stated simply, the observable event set $\Sigma_o \subseteq \cup_{i=1}^{n}\Sigma_i$ selected in model abstraction must satisfy three conditions [30, 33]:

1. *Shared observability*: all events common to two or more components must be observable, i.e.,

   $$(\forall i,j \in \{1,\cdots,n\})i \neq j \Rightarrow \Sigma_i \cap \Sigma_j \subseteq \Sigma_o$$

   This condition ensures the compositionality of model abstraction.
2. *Natural observer property*: let $L_i \subseteq \Sigma_i^*$ be the marked language of subsystem $i(=1,\cdots,n)$. The natural projection $P_i : \Sigma_i^* \to (\Sigma_i \cap \Sigma_o)^*$ is a natural observer if

   $$(\forall t \in P_i(L_i))\big(\forall s \in \overline{L_i}\big)P_i(s) \leq t \Rightarrow \big(\exists u \in \Sigma_i^*\big)su \in L_i, P_i(su) = t$$

   The set Σ_o must be large enough so that what we expect (t) in the abstracted model ($P(L_i)$) is surely realizable (su) in the original subsystem (L_i). This is important for non-blocking control.
3. *Output control consistency* (OCC): for every string $s \in \overline{L_i}$ of the form

   $$s = \sigma_1 \cdots \sigma_k \text{ or } s = s'\sigma_1 \cdots \sigma_k,\ k \geq 1$$

   where s' terminates with an observable event in $\Sigma_o, \sigma_j \in \Sigma_i - \Sigma_o(j=1,\cdots,k-1)$ and $\sigma_k \in \Sigma_o$, it must be true that

   $$\sigma_k \in \Sigma_{uc} \Rightarrow (\forall j \in \{1,\cdots,k\})\sigma_j \in \Sigma_{uc}$$

   According to this requirement, if an uncontrollable event is observable, all its immediately preceding controllable events must be observable as well. This property enables us to make the right control decision to prevent the uncontrollable event from occurring in the abstracted model. The condition ensures the maximal permissiveness of control design.

For details about the three sufficient conditions, see [30, 31, 33, 34]. The conditions will be illustrated for the production cell example below, and can be established by software tools [43, 45]. With these tools designers may follow the steps in the example to solve their own supervisory control problems. The tool we use is XPTCT [45].

4 Production cell

This section describes a production cell [15, 46] that will be used to exemplify the proposed approach. Overall cell operation should be clear from Fig. 2.

The cell is made up of seven components: stock, feed belt, elevating rotary table, robot, press, deposit belt, and crane. Each executes its own actions asynchronously and independently, apart from the synchronization needed for cooperation and safety. The cell processes workpieces, called ‘blanks’, as follows:

1. The stock inputs blanks to the system on the feed belt.
2. The feed belt forwards a blank to the elevating rotary table.
3. The table lifts and rotates the blank to the position where arm1 of the robot picks it up.
4 Arm1 retracts/extends its length while the robot rotates to the press, so that arm1 can transfer the blank to the press.
5. The blank is forged by the press.
6. The forged blank is picked up by arm2 of the robot.

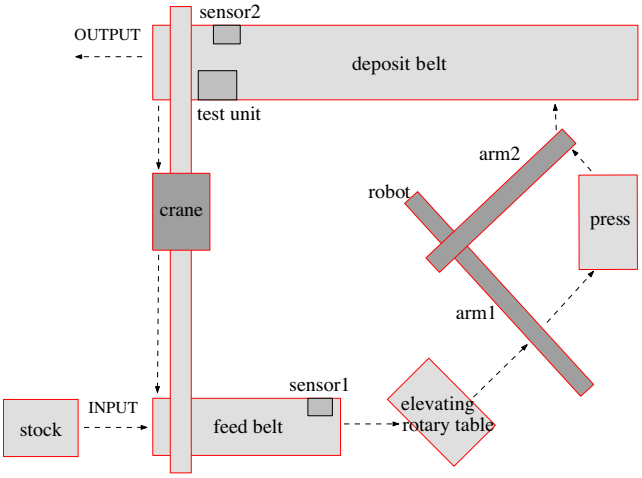


Fig. 2 System structure of production cell

7. Arm2 retracts/extends its length while the robot rotates to the deposit belt, so that arm2 can transfer the forged blank to the deposit belt.
8. The deposit belt forwards the blank to an end station, where a test unit checks if the forging was successful.
9. If the blank passes the test, it will be output from the system; otherwise, it is picked up by the crane and moved to the feed belt for another forging.

From the above description, one can identify problems the system may encounter, including underflow/overflow of buffers, collision between robot arms and table/press, and deadlock due to `choking' of the loop by which the crane feeds back blanks that tested faulty. In the next section, we examine the detailed behavior of each component, and establish the constraints needed on component activities.

Ma and Wonham have solved this problem using the *state tree structure* (STS) [15], but did not elaborate on the control logic of the solution; the logic is presented in binary decision diagrams (BDD) which may not be transparent for the average designer. Moreover, the component models in [15] do not always clearly separate plant from specification, as is normal practice in control theory. Thus in SCT, plant components typically behave independently and concurrently, except when interacting and collaborating with other components in accordance with specification. Our proposed approach to the production cell follows this principle more uniformly and, we think, exhibits the control logic more transparently. While our system models for the production cell differ somewhat from those in [15], they do, however, lead to the same system behavior.

5 System modeling

This section describes the plant and specification models of both individual components and the overall system.

Table 1 Event in stock

Event	Controllable	Description
blank_add	Y	Put a blank on feed belt

5.1 Stock

The stock adds blanks to the cell by placing them on the feed belt. It generates only one controllable event: *blank_add*, as explained in Table 1 and shown in Fig. 3[1].

5.2 Crane

The crane picks up a faulty blank from the deposit belt (event *Cr_mOn*) and transfers it back to the feed belt. As the deposit belt is assumed to be lower than the feed belt, the crane has to traverse vertically (*Cr_U, Cr_66, Cr_SVf*) as well as horizontally (*Cr_2FB, Cr_FB, and Cr_SHf*). On reaching the feed belt, the crane deposits the blank (*Cr_mOff*), then returns to the deposit belt by the reverse path (*Cr_2DB, Cr_DB, Cr_SHf, Cr_D, Cr_95, Cr_SVf*). These crane events are listed in Table 2. We model the crane as the two automata in Fig. 4. Automaton **Cr_V** describes the vertical activity of the crane and automaton **Cr_H** the horizontal activity. To represent the complete behavior of the crane in one automaton model, we compute the *synchronous product* $\mathbf{Cr} := \mathbf{Cr\_V} \| \mathbf{Cr\_H}$. The computation can be carried out by command *sync* of XPTCT [45] software.

5.3 Feed belt

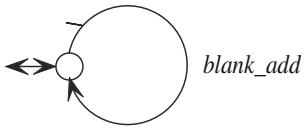
The feed belt transfers blanks to the elevating rotary table. Once loaded with a blank, the feed belt moves (*FB_F*) it toward the table. Sensor1 at the end of the belt will switch to 'on' (*FB_s1On*) when it detects the blank's arrival. The blank will be placed on the table (*FB_tau*) if the table is ready to accept it; otherwise the feed belt must stop (*FB_Sf*) until the table becomes available. Sensor1 switches to 'off' (*FB_s1Off*) when the blank leaves. These events are listed in Table 3 and the corresponding state transitions shown in Fig. 5.

The feed belt must synchronize with the stock and the crane to feed in blanks, subject to the following two specifications:

- The feed belt can hold at most two blanks to prevent collision. This specification is enforced by automaton **FB1** in Fig. 6.

[1] In transition diagrams, by convention, we mark controllable events with a tick on the corresponding arrows.

Fig. 3 Plant model of stock



- If there is already one blank on the feed belt, then before it reaches the end of the belt and activates sensor1 (*FB_s1On*), a new blank is prohibited from being loaded, again to avoid collision. This is enforced by automaton **FB2** in Fig. 6.

FB1 is the model of a buffer with capacity 2, whose content is incremented by events *Cr_mOff, blank_add*, and decremented by event *FB_s1Off*. Likewise **FB2** is the model of a buffer with capacity 1, incremented by events *Cr_mOff, blank_add*, and decremented by event *FB_s1On*. In fact many control specifications amount to just the synchronous product of buffers.

FB1 and **FB2** prescribe the interaction among plant models **Cr, Stock, FB**, as in the diagram[2], Fig. 7. Here a block stands for a plant component and an oval for a control specification. An edge connects a block and an oval if and only if the corresponding plant component and control specification share a common event. If the specification model is a buffer and the common event increases its content, then an arrow enters the specification, namely, the plant component at the tail of the arrow feeds the buffer. If the common event decreases the buffer's content, then an arrow leaves the specification, namely, the plant component at the head of the arrow unloads the buffer. For illustration, **Stock** shares event *blank_add* with **FB1, FB2** and this event increases the contents of both buffers as in Fig. 6.

5.4 Elevating rotary table

The table receives blanks from the feed belt and provides them to arm1 of the robot. Since the feed belt is located lower than arm1, the table, upon receiving a blank from the feed belt, must move up (*Ta_U, Ta_T, Ta_STf*) and turn to an appropriate angle (*Ta_R, Ta_50, Ta_S50f*) for arm1 to pick up the blank. After this, the table moves down (*Ta_D, Ta_B, Ta_SBf*) and returns to the position for receiving new blanks from the feed belt (*Ta_L, Ta_0, Ta_S0f*). The event list for the table is presented in Table 4 and the models are in Fig. 8.

The control specification for the table serves to coordinate the actions of the table, the feed belt, and arm1 of the robot, as shown in Fig. 9. By **Ta1**, the feed belt cannot transfer a blank to the table (*FB_s1Off*) until the table is at the bottom position (*Ta_SBf*). Only after the occurrence of *FB_s1Off* is the table allowed to move up (*Ta_U*). By **Ta2**, arm1 cannot pick up the blank from the table (*A1_mOn*) until the table stops at the top position (*Ta_STf*). Only after the occurrence of *A1_mOn* can the table move down (*Ta_D*). Similarly, **Ta3, Ta4** serve to synchronize the rotating actions of the table with events of the feed belt and arm1.

Figure 10 is the interconnection diagram showing the relationship among the table, the feed belt, and arm1 of the robot. The interpretation of the blocks, ovals, and edges in the diagram is the same as in Fig. 7. Since none of the four models is a buffer, there is no arrow in this diagram.

[2] With further assumptions on plant and specification models, the diagram is called a *control-flow net* [31, 33].

Table 2 Events in crane

Event	Controllable	Description
Cr_mOn	Y	Pick up one blank
Cr_mOff	Y	Put down one blank
Cr_U	Y	Move up
Cr_66	N	Height matches fb
Cr_D	Y	Move down
Cr_95	N	Height matches db
Cr_SVf	N	Stop vertical move
Cr_2FB	Y	Move to fb
Cr_FB	N	Reach fb
Cr_2DB	Y	Move to db
Cr_DB	N	Reach db
Cr_SHf	N	Stop horizontal move

5.5 Press

The press forges blanks; it has three main positions: bottom, middle, and top. The press is loaded by arm1 at the middle position, unloaded by arm2 at the bottom, and forges the blank at the top. The press ascends from bottom to middle (*Pr_UB, Pr_MU, Pr_SMf*) and, after being loaded by arm1, continues to top (*Pr_UM, Pr_T, Pr_STf*), where it forges a blank. Upon finishing, the press descends to the bottom (*Pr_D, Pr_MD, Pr_B, Pr_SBf*) and prepares to unload. Table 5 lists the events of the press and Fig. 11 shows its plant model.

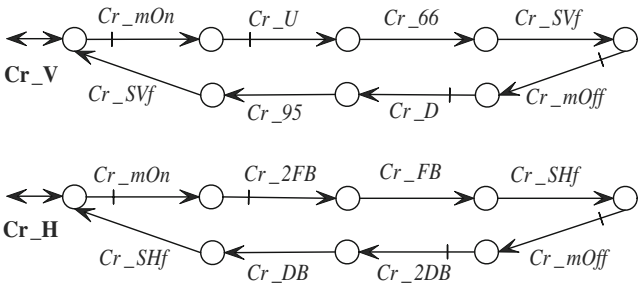


Fig. 4 Plant model of crane

Table 3 Events in feed belt

Event	Controllable	Description
FB_F	Y	Move forward
FB_Sf	N	Stop
FB_tau	Y	Move forward one step
FB_s1On	N	Sensor1 is on
FB_s1Off	N	Sensor1 is off

The press synchronizes with arm1 and arm2 of the robot. Arm1 may place one blank on the press (*A1_mOff*) only if the press is at the middle position (*Pr_SMf*). Before loading is completed, the press may not move up (*Pr_UM*). This specification is realized by automaton **Pr1** in Fig. 12. Moreover, arm2 may pick up a blank from the press (*A2_mOn*) only if the press is at the bottom position, and the press must not move up before a pickup (*Pr_UB*). **Pr2** in Fig. 12 enforces this specification.

Note that unlike all other automata in the paper, the marker state of **Pr2** is not the initial state, because when the robot and its arms are at their initial states, as will be shown in Fig. 17 shortly, **Pr2** must reach the marker state in Fig. 12. If we allow the occurrence of event *A2_mOn*, arm2 needs a further sequence of actions to reach its marker state, and these actions require precise collaboration with arm1, the rotary base, the press, and the rotary table, so that they cannot all reach the marker states simultaneously. Therefore, if the marker state of **Pr2** were set to be its initial state, non-blocking control could not be realized.

The interconnection between the two arms and the press is shown in Fig. 13. Automata **A1, A2** in the diagram will be displayed later in Fig. 17.

5.6 Deposit belt

The deposit belt transfers blanks (*DB_F*) loaded from arm2 of the robot to the other end, where Sensor2 will switch to 'on' (*DB_s2On*) if it detects the arrival of a blank, and 'off' (*DB_s2Off*) to indicate the blank is being checked by the test unit (*DB_Sf*). If the blank passes the check (*DB_yes*), then it will be output from the system (*FB_tau*); otherwise (*DB_no*) it will be picked up by the crane. Table 6 lists the events of the deposit belt and Fig. 14 presents the plant model.

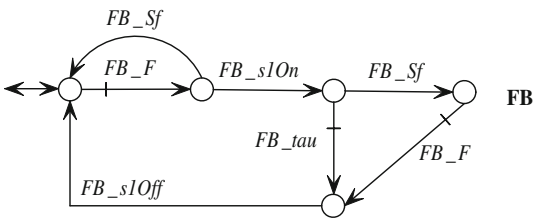


Fig. 5 Plant model of feed belt

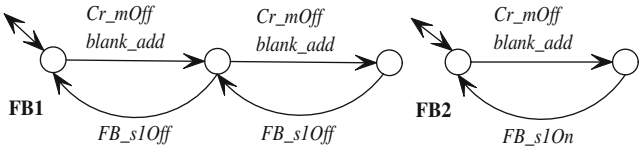


Fig. 6 Specification model of feed belt

Event *DB_Ff* is the only controllable event of **DB**. Note the difference between events *DB_Ff* and *DB_tau*, whose counterpart is *FB_tau* for the feed belt. Event *DB_Ff* represents a command to move the deposit belt. After the occurrence of this event, the deposit belt keeps on moving until event *DB_Sf* occurs. By contrast, event *DB_tau* represents a momentary moving command, so that the deposit belt will stop just long enough to deposit a blank at the end.

Three specifications constrain the activity of the deposit belt. First, the deposit belt should hold at most two blanks at the same time in order to prevent collision. By **DB1** in Fig. 15, it is a buffer of capacity 2. Second, if there is already one blank on the deposit belt, then a new one cannot be loaded until the first reaches the other end of the belt and is detected by sensor2. Its model is **DB2** in Fig. 15 and is a buffer of capacity 1. Third, if a blank fails the check, then it must be taken up by the crane rather than left in the system. Its model is **DB3** in Fig. 15 and is also a buffer of capacity 1. The interconnection diagram for **DB** is Fig. 16, where automaton **A2** is the plant model of arm2, to be introduced in the next section.

5.7 Robot

The robot consists of a rotary base and two orthogonal arms. Each arm can extend and retract, and can load or unload a blank by turning its magnet on or off. Arm1 transfers blanks from the table to the press and arm2 from the press to the deposit belt. Table 7 lists the events of the robot. The operational cycle of the rotary base is as follows. Initially it stays at 50°, then rotates left to 35° (*Ro_L, Ro_35, Ro_S35*), −90° (*Ro_L, Ro_−90, Ro_S−90*). The base then switches direction and rotates right, returning to 50° (*Ro_R, Ro_50, Ro_S50*) in one step. Automaton **Ro** in Fig. 17 models the rotary base.

Synchronizing with the rotary base at its initial length, arm1 picks up a blank from the table (*A1_mOn*), then either

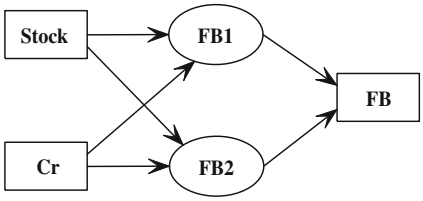


Fig. 7 Interconnection diagram of feed belt

Table 4 Events in elevating rotary table

Event	Controllable	Description
Ta_U	Y	Move up
Ta_T	N	Reach top
Ta_STf	N	Stop at top
Ta_D	Y	Move down
Ta_B	N	Reach bottom
Ta_SBf	N	Stop at bottom
Ta_R	Y	Turn right
Ta_50	N	Reach 50°
Ta_S50f	N	Stop at 50°
Ta_L	Y	Turn left
Ta_0	N	Reach 0°
Ta_S0f	N	Stop at 0°

extends directly to 65 (*A1_F, A1_S65*), or first retracts to 37 (*A1_B, A1_37, A1_S37*), to avoid collision, and then extends to 65. After placing the blank on the press (*A1_mOff*), it retracts to 52 (*A1_B, A1_52, A1_S52*) for the next operational cycle. Automaton **A1** in Fig. 17 models arm1.

Initially arm2 has length 0, as shown by automaton **A2** in Fig. 17. To start a cycle, it extends to 80 (*A2_F, A2_80, A2_S80*) and picks up a forged blank from the press (*A2_mOn*). Then it retracts to 57 (*A2_B, A2_57, A2_S57*) and places the blank on the deposit belt (*A2_mOff*). Two possible actions may follow. It either first retracts to 0 (*A2_B, A2_0, A2_S0*), to avoid collision, and then extends to 80 or directly extends to 80. The (unrestricted) system behavior of the robot is the synchronous product of the rotary base, arm1 and arm2.

The rotary base and the two arms must cooperate in the following ways.

- Arm1 may pick up a blank from the table only when the base is at 50°, as enforced by **R1** in Fig. 18.
- Arm2 may pick up a forged blank from the press only at 35°, except for the first time that the robot reaches 35°. This is because initially there is no blank at the press. The specification is enforced by **R2** in Fig. 18.
- The robot may turn left only after either arm picks up a blank, as enforced by **R3** in Fig. 18.

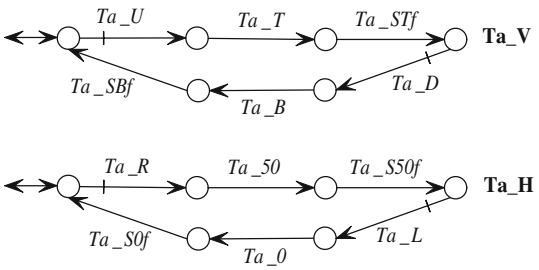


Fig. 8 Plant model of elevating rotary table

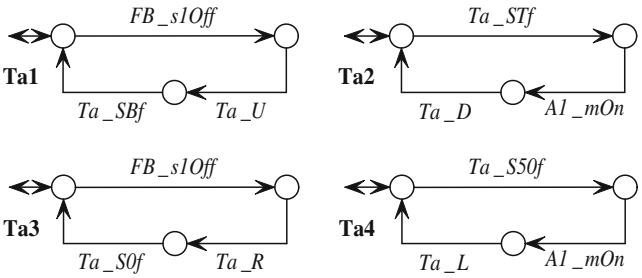


Fig. 9 Specification model of elevating rotary table

- Arm1 and arm2 may place a blank on the press or the deposit belt, respectively, only when the base is at −90°; and only after both loading actions are completed can the robot turn right. This is realized by **R4**, **R5** in Fig. 18. Similar to **R2**, arm2 does nothing when the robot first reaches −90°.

Figure 19 shows the interconnection relationship among the base and two arms.

5.8 Additional specifications for collision avoidance

This section formalizes three mutual exclusion restrictions to avoid collision, namely, certain states of related plant models cannot be occupied simultaneously. The first specification is to prevent collision between arm1 and the press, which could happen when arm1 is longer than 37, the robot is at −90°, and the press is at the top position. **A1P** in Fig. 20 enforces this specification. Since the two transitions pointing to the right have the identical event labels, we only show the event label once to avoid clutter. The two transitions pointing leftwards are labeled the same way. According to plant model **A1** in Fig. 17, arm1 has length 52 at its initial and marker state. Therefore, the initial and marker state of **A1P** is the state in the middle, meaning events *Ro_−90, Pr_T* cannot both happen before arm1 retracts to safe length (*A1_37*).

The second specification is to prevent collision between arm2 and the press, which could occur when arm2 has length greater than 0, the robot is at 35°, and the press is not at the bottom position. **A2P** in Fig. 20 enforces this specification. Similar to **A1P**, transitions with the same directions share the same labels. Referring to the plant models of the robot in Fig. 17 and the press in Fig. 11, we set the leftmost state to be the initial and marker state.

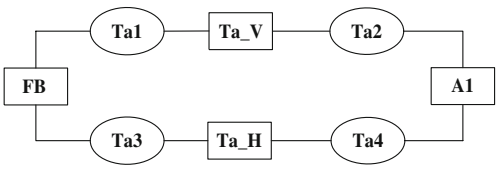


Fig. 10 Interconnection diagram of table

Table 5 Events in press

Event	Controllable	Description
Pr_UB	Y	Ascend from bottom
Pr_MU	N	Ascend to middle
Pr_SMf	N	Stop at middle
Pr_UM	Y	Ascend from middle
Pr_T	N	Reach top
Pr_STf	N	Stop at top
Pr_D	Y	Descend
Pr_MD	N	Descend to middle
Pr_B	N	Reach bottom
Pr_SBf	N	Stop at bottom

The last specification is to prevent collision between arm1 and the table, and is more complex than the previous two situations, where collision occurs if both the robot and the press reach dangerous positions together. A collision between arm1 and the table, however, depends not only on the positions of the robot and the table but also on whether arm1 is holding a blank. If arm1 has just picked up a blank from the table and has not yet retracted, the robot stays at 50°, and the table returns to the top position again with a blank, then the two blanks touch each other, which is considered a collision.

This specification should prevent the table from returning to the top position, before the loaded arm1 retracts to a safe distance or the robot turns away from the table. The model of this specification is automaton **A1T** in Fig. 20. Event *Ta_T* is disabled at the state at the right side; i.e., before the loaded arm1 rotates from 50° to 35°, the table is not allowed to reach the top. Note that we do not allow the table to return to the top even if arm1 retracts to the safe length 37. This is because the loaded arm1 can first retract to length 37 and then extend to length 65 even though the robot stays at 50°. During this process, if we allow the table to return to the top position after arm1 retracts to length 37, collision will occur when the arm extends again.

6 Interconnection diagram of production cell

Finally, in Fig. 21 we draw the interconnection diagram of the production cell. This is simply a combination of the

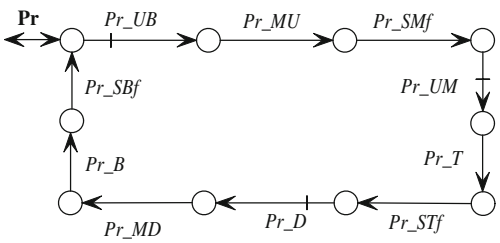


Fig. 11 Plant model of press

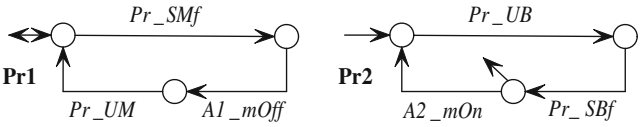


Fig. 12 Specification model of press

partial diagrams in Figs. 7, 10, 13, 16, and 19, together with the three specification models in Fig. 20. Since the diagram is too complex to be reduced by the graphical methods in [31, 33], we shall find the non-blocking control for this example using the computational methods based on model abstractions [30]. The whole net is divided into two subsystems by the dashed lines in Fig. 21. Subsystem 1 includes plant components **DB**, **Cr**, **Stock**, **FB**, **Ta_V**, **Ta_H**, and specifications **DB3**, **FB1**, **FB2**, **Ta1**, **Ta3**; subsystem 2 includes plant components **A1**, **A2**, **Ro**, **Pr**, and specifications **Pr1**, **Pr2**, **A1P**, **A2P**, **R1**, ..., **R5**. The two subsystems connect through specifications **DB1**, **DB2**, **A1T**, **Ta2**, **Ta4**. In the following sections, we design decentralized supervisors for each individual specification and check to see if these decentralized supervisors result in conflict. In that case, coordinators will be designed as well.

7 Decentralized supervisors

This section corresponds to Step 1 of the computational process presented in Sect. 3. We first simplify the plant model based on the events present in specifications and on model abstraction principles. For example, according to Fig. 21, **Cr** is related to specifications **DB1, DB2** via *Cr_mOn*, and to **FB1, FB2** via *Cr_mOff*. Other events in Table 2 are irrelevant to the control synthesis and should be concealed for model abstraction. By algorithms in [43] and [41], event set {*Cr_mOn, Cr_mOff*} defines a natural observer with OCC property for **Cr_V** and **Cr_H** in Fig. 4. The significance of the natural observer and OCC were described informally at the end of Sect. 3. By Proposition 4.5 in [33], the natural projection is also an observer with OCC property for **Cr** = **Cr_V**||**Cr_H**. Its abstraction is simply the buffer with size 1 in Fig. 22. This computation and those that follow were performed with XPTCT software [45]. In the future we shall use the abstraction and call it **Cr**.

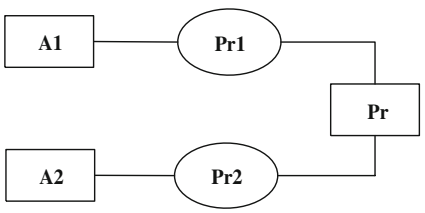


Fig. 13 Interconnection diagram of press

Table 6 Events in deposit belt

Event	Controllable	Description
DB_F	Y	Move forward
DB_Sf	N	Stop
DB_tau	N	Move forward one step
DB_s2On	N	Sensor2 is on
DB_s2Off	N	Sensor2 is off
DB_yes	N	Blank acceptable
DB_no	N	Blank not acceptable

Similarly, we simplify the plant models **Ta_V**, **Ta_H** to those in Fig. 23, **Pr** to that in Fig. 24, and **A1, A2** to those in Fig. 25.

Next, we find decentralized supervisors for the individual specifications by standard SCT algorithms. In XPTCT [45], we use command *Supcon* to find an optimal and nonblocking supervisor and command *Supreduce* to reduce it. The reduced supervisors [47] of most specifications are simple and shown in Figs. 26, 27, 28, 29, 30, and 31, except those for **DB1**, **A1P** and **A2P**, which are too complicated to show here. The control logic of the three decentralized supervisors, however, is still evident.

The supervisor **DB1C** for **DB1** has seven states. The result was surprising, because it seems adequate simply to disable event *DB_Ff* at the initial state of **DB1** in Fig. 14. On a closer look at the system, however, the correct control logic turns out to be more subtle. For example, when *A2_mOff* occurs once and event *DB_Ff* occurs, we must disable event *Cr_mOn*; otherwise uncontrollable event *DB_tau* may occur and underflow will result because of the uncontrollable transitions between events *DB_Ff* and *DB_tau*.

The supervisors **A1PC**, **A2PC** for **A1P**, **A2P** have nine and eight states, respectively. Similar to **DB1C**, their complexities are also due to the uncontrollable transitions directed to the right. For instance, at the initial state of **A1P** of Fig. 20, when event *Pr_UM* occurs, event *Ro_−90* must be preempted by disabling event *Ro_L*.

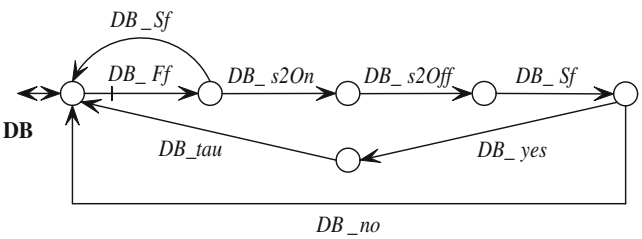


Fig. 14 Plant model of deposit belt

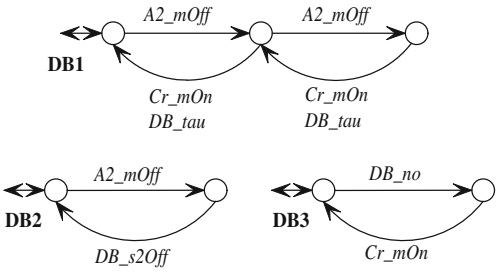


Fig. 15 Specification model of deposit belt

8 Subsystem supervision

Step 2 is to decompose the whole system into proper subsystems according to the interconnection relation. As shown in Fig. 21, the production cell is divided into two subsystems. The next step, Step 3, is to check if each of the two subsystems is non-blocking under the decentralized supervisors. The controlled behavior of a system is the synchronous product of all the plant components and the decentralized supervisors. This computation is realized by command *Sync* in XPTCT. We call the model of subsystem 1 **SUB1**; it has 2,478 states and is non-blocking. The model of subsystem 2 is **SUB2**; it has 820 states and unfortunately is blocking.

We find a coordinator for **SUB2** using the *Supcon* and *Supreduce* commands. We use the *Supcon* command to eliminate blocking states while preserving controllability. The result of this command is **SUB2S**and has 650 states. Amazingly, the result of the *Supreduce* command is automaton **SUB2C** with only 3 states[3]. Its state transition diagram is shown in Fig. 32.

The only control action of this coordinator is to disable event *A2_F* at the initial state. This corresponds to the following two logic rules.

- While the robot first rotates away from its initial position 50° to −90°, arm2 must stay put at its initial state; otherwise, it may result that the robot is at −90°, the press is at the middle position, and arm2 overhangs the deposit belt with length 80. This gives rise to deadlock, as arm2 cannot retract to length 0 because event *A2_mOn* cannot occur in the first rotation. The press cannot return to the bottom position because, by models **Pr** in Fig. 24 and **Pr1C** in Fig. 28, the press must move up to the top position before it moves down. Moving up is, however, prohibited by **A1PC** (see Fig. 20), as it would cause collision between arm1 and the press. Finally, to avoid the collision between arm2

[3] The correctness of a *Supreduce* result can be independently verified in XPTCT using other commands [41]: first compute the synchronous product of the plant and the reduced supervisor, then check if the result is isormorphic to the *Supcon* result.

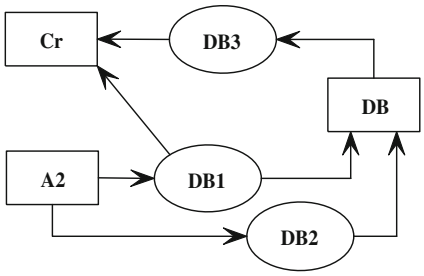


Fig. 16 Interconnection diagram of deposit belt

and the press, the robot cannot turn right because of **A2PC** (see Fig. 20).

- The dashed transition *A2_F* in **A2** must always be disabled; otherwise the deadlock situation mentioned in rule 1 will arise again. Each time the robot rotates away from −90° to 50°, the press moves to the bottom position. The sequence *Ro_50.Pr_B* is then detected and **SUB2C** reaches the state at the right-hand side of the diagram. In the subsequent rotation, the robot rotates left to 35° and arm2 extends to pick up a blank from the press. Then sequence *Ro_L.A2_F* or *A2_F. Ro_L* is detected and **SUB2C** returns to the initial state. The robot proceeds to rotate left to −90° via *Ro_L*. Then arm2 places a blank on the deposit belt via *A2_mOff* and must retract because event *A2_F* is disabled at the initial state of **SUB2C**.

Table 7 Events in robot

Event	Controllable	Description
Ro_L	Y	Robot turns left
Ro_R	Y	Robot turns right
Ro_35	N	Robot reaches 35°
Ro_S35	N	Arm2 reaches press
Ro_−90	N	Robot reaches –90°
Ro_S−90	N	Arm1 reaches press and arm2 reaches db
Ro_50	N	Robot reaches 50°
Ro_S50	N	Arm1 reaches table
A1_mOn	Y	Arm1 picks up a blank
A1_mOff	Y	Arm1 unloads a blank
A1_B	Y	Arm1 retracts
A1_F	Y	Arm1 extends
A1_37	N	Arm1 reaches 37
A1_S37	N	Arm1 reaches safe extension
A1_65	N	Arm1 reaches 65
A1_S65	N	Arm1 reaches press
A1_52	N	Arm1 reaches 52
A1_S52	N	Arm1 reaches table
A2_mOn	Y	Arm2 picks up a blank
A2_mOff	Y	Arm2 unloads a blank
A2_B	Y	Arm2 retracts
A2_F	Y	Arm2 extends
A2_0	N	Arm2 reaches 0
A2_S0	N	Arm2 reaches safe extension
A2_80	N	Arm2 reaches 80
A2_S80	N	Arm2 reaches press
A2_57	N	Arm2 reaches 57
A2_S57	N	Arm2 reaches db

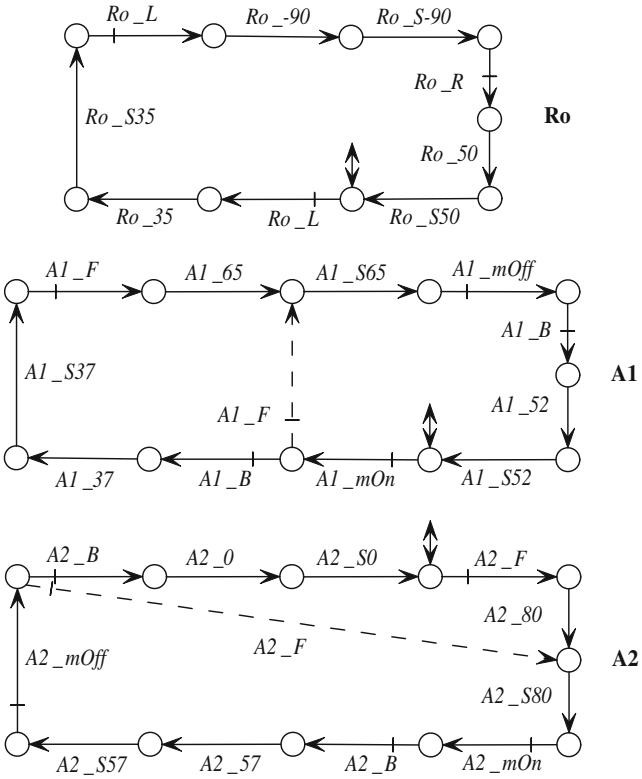


Fig. 17 Plant model of robot

9 Abstractions of the subsystems

This section describes Step 4 to obtain useful model abstractions of the subsystems. The two subsystems do

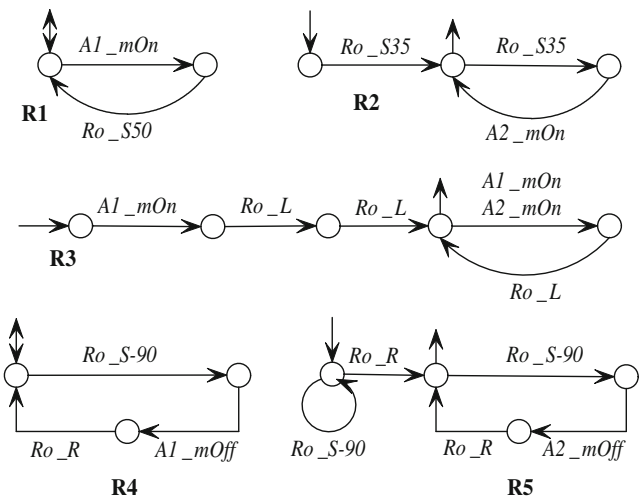


Fig. 18 Specification model of robot

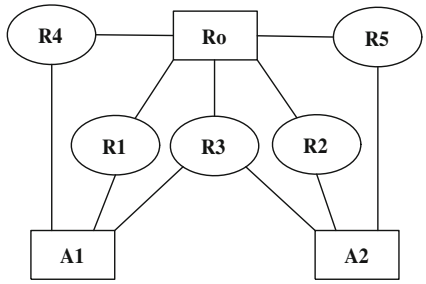


Fig. 19 Interconnection diagram of robot

not directly share joint events, but each does with supervisors **DB1C**, **DB2C**, **Ta2C**, **Ta4C**, and **A1TC**, as shown in Fig. 21. The joint events are listed in Table 8. The last row in the table shows the common events of the two subsystems; these must be made observable for further verification.

To ensure the OCC property, we first add event *Ta_R* to the observable event set of **SUB1**, as it is the immediately preceding controllable event of *Ta_S50f*, and add event *Ro_L* to the observable event set of **SUB2S**, as it is the immediately preceding controllable event of *Ro_35*.

Next we obtain the observers for the two subsystems. Using our algorithm in [43] or Chapter 5 of [33], we enlarge the two observable event sets to

$$\Sigma_1 = \left\{ \begin{array}{l} blank\_add, Cr\_mOn, Cr\_mOff, Ta\_U, Ta\_STf, \\ Ta\_D, Ta\_R, Ta\_S50f, Ta\_L, DB\_Ff, DB\_Sf, \\ DB\_s2On, DB\_s2Off, DB\_yes, DB\_no, DB\_tau \end{array} \right\}$$

and

$$\Sigma_2 = \{A1\_mOn, A2\_mOff, A2\_F, Ro\_L, Ro\_35\}$$

The natural projections defined on these two observable event sets are indeed observers and OCC.

Using the two natural projections (*Project* command of XPTCT), we find the model abstractions of the two subsystems as **PSUB1** of 644 states and **PSUB2S** of 13 states. Recall that the original state sizes of the two non-blocking subsystems were 2478 and 650. Thus the abstractions are quite economical.

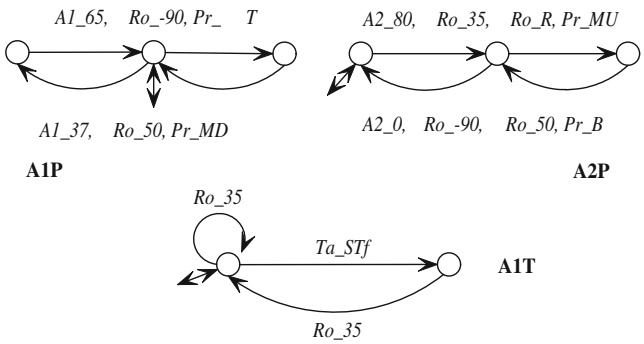


Fig. 20 Specifications for collision avoidance

10 Overall system coordination

At Step 5, we investigate if the synchronous product of the two model abstractions is non-blocking. If so, a result in [30, 33] guarantees that the whole system is non-blocking under the decentralized supervisors achieved above. Because the model abstractions are significantly smaller in state size than the original subsystems, we can check for system non-blocking with much less computational effort.

We compute the synchronous product of the two model abstractions and the five connecting supervisors: **DB1C**, **DB2C**, **Ta2C**, **Ta4C**, and **A1TC**. The result is automaton **SYS** with 6367 states, and is blocking.

To obtain non-blocking control, we take **SYS** as plant and use the *Supcon* command to find an optimal and non-blocking control for **SYS**. The result is a non-blocking automaton **NBSYS** of 6250 states. Owing to the large number of states, it is very time-consuming to simplify **NBSYS** by the *Supreduce* command. Alternatively, we shall design a coordinator to resolve blocking in **SYS** by inspection of the cell's mode of operation.

We note that the cell contains a loop with material feedback, of faulty blanks. New blanks enter the cell via event *blank_add* and completed blanks leave via event *DB_tau*. The model of **DB** in Fig. 14 shows that the occurrence of *DB_tau* or *DB_no*, returning a faulty blank for reforging, is uncontrollable. To accommodate such a blank, there must be at least one empty slot in the cell; Otherwise, if *DB_no* occurs, the cell will deadlock[4].

The control logic of the coordinator is therefore to maintain at least one empty slot in the cell. To formalize it, we calculate the maximal number of blanks that the cell may hold. By **Cr** in Fig. 22, the crane can hold one blank; by model **FB1** in Fig. 6, the feed belt can store two blanks; by Fig. 9, the rotary elevating table can store one blank; by Fig. 25, arm1 and arm2 can each hold one blank. Although the press can also hold one blank, it does not contribute to the cell capacity, because if a blank is in the press, either arm1 or arm2 must be free. By model **DB1** of Fig. 15, the deposit belt contains at most two blanks. Thus, in total, the cell capacity is eight.

[4] This is nothing but an instance of a feedback loop going unstable if it is "driven too hard", namely "its gain is too high". Here "instability" (deadlock) is the inability of the system to return to its "equilibrium" (initial) state.

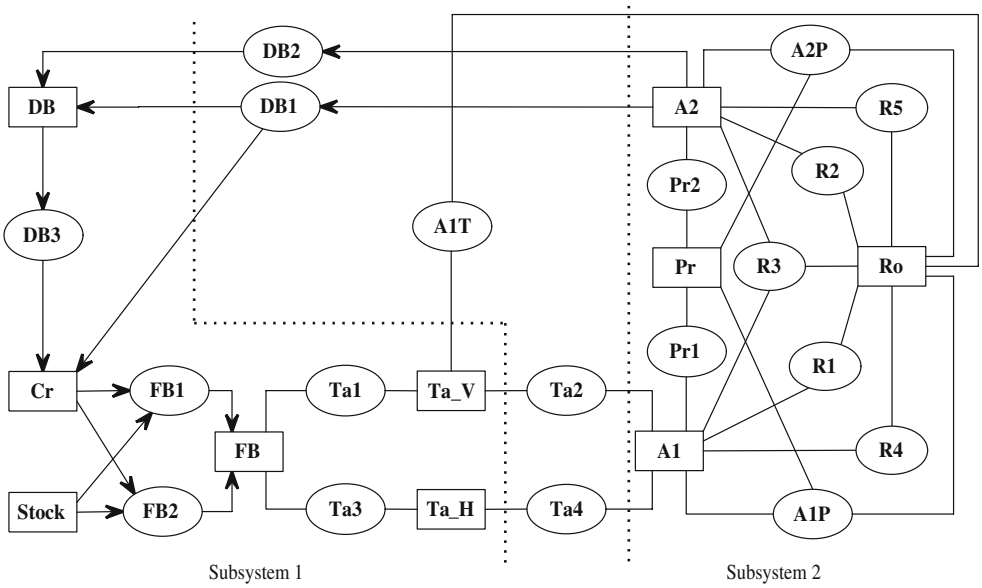


Fig. 21 Interconnection diagram of production cell

The control logic to prevent blocking of the decentralized control is therefore to disable event *blank_add* when there are seven blanks in the production cell. It is implemented by automaton **TOTAL** in Fig. 33. To test its correctness, we compute the synchronous product of **SYS** and **TOTAL**, resulting in a non-blocking system with 6,211 states. The model **TOTAL**, therefore, is a non-blocking but not optimal coordinator to **SYS**, as the optimal non-blocking supervisor **NBSYS** has 6,250 states.

This result does suffice for non-blocking control, but we shall look further into why it is not maximally permissive, and look for a non-blocking coordinator that is actually optimal. The secret lies in model **DB** of the deposit belt in Fig. 14. After the occurrence of *DB_yes*, event *DB_tau* will occur inevitably and *DB_no* cannot occur. Consequently, if there are already seven blanks in the production cell and then *DB_yes* occurs, we may still permit a new blank to enter the cell via *blank_add*, because an empty slot will soon be created and no more blanks may enter the cell before that. This favorable situation is not recognized, however, by **TOTAL**.

In this light, we improve coordinator **TOTAL** to **NTOTAL**, as in Fig. 34. The synchronous product of **SYS** and **NTOTAL** is isomorphic to the non-blocking system **NBSYS**. Thus we have confirmed that **NTOTAL** is an optimal and non-blocking coordinator for the decentralized supervisors of the production cell.

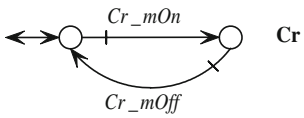


Fig. 22 Simplified plant model of crane

11 Conclusions

We have obtained a maximally permissive and non-blocking supervisor for the production cell using a structured supervisor design approach. With the proposed approach, the design requires smaller computational resources and can even be realized by the pedagogical design software XPTCT, which was thought impossible before. The supervisory control takes the form of a group of 21 simple decentralized supervisors with state sizes ranging from 2 to 15, whose control actions can be readily translated into 'if ... then ...' logic rules which are readily understandable to the designer.

The main contribution of the proposed structural approach is the sufficient conditions, i.e., natural observer and OCC, to design proper model abstractions of relatively complex systems, so that we can flexibly integrate decentralized and hierarchical control architectures. Computational tools have also been provided to check these sufficient conditions and, if necessary, arrange that they hold.

We do not claim that the proposed approach is computationally the most efficient possible, as for instance computational methods based on BDD and State Tree Structures [15] are more powerful. No doubt bringing in these methods would be advantageous. We also recognize that the proposed approach is by no means fully automated, and probably could not be made to be. Indeed, successful design will usually call on expert knowledge both of DES modeling and of the physical system to be controlled. Nevertheless, our theory and computational tools provide the designer with strong support: the control logic of the decentralized supervisors is transparent, and the modular structure admits easy implementation.

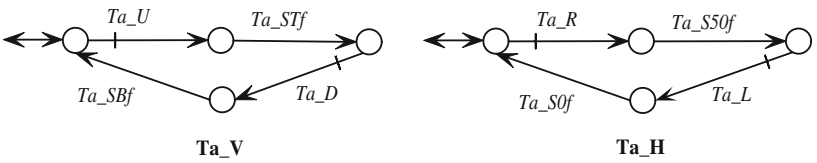


Fig. 23 Simplified plant model of table

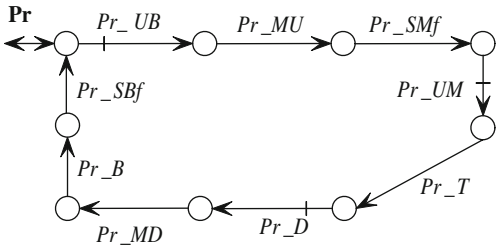


Fig. 24 Simplified plant model of press

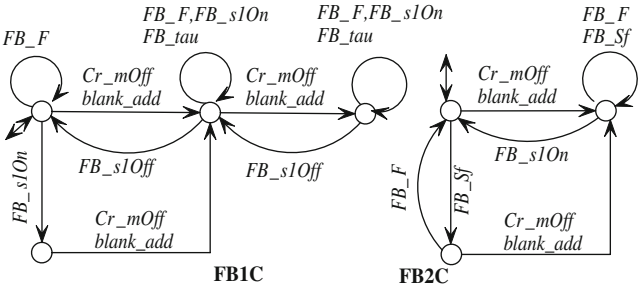


Fig. 26 Supervisors of **FB1**, **FB2**

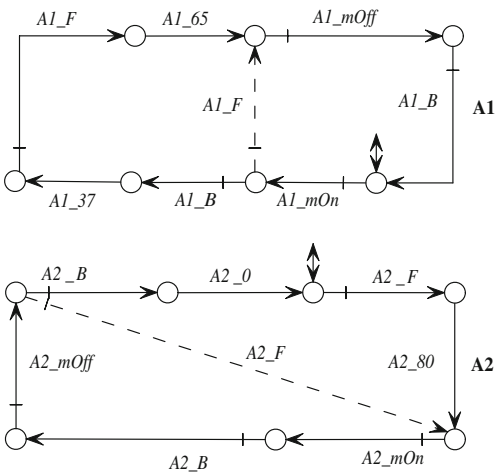


Fig. 25 Simplified plant models of arms

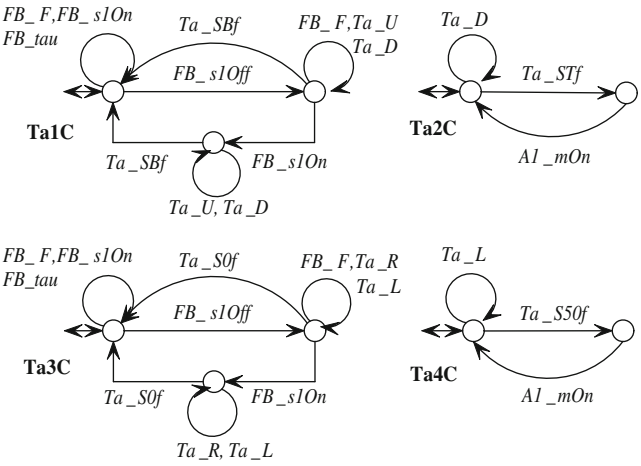


Fig. 27 Supervisors of **Ta1** to **Ta4**

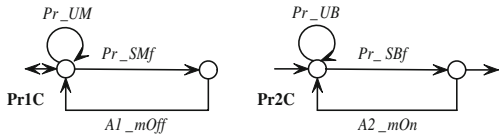


Fig. 28 Supervisors of **Pr1**, **Pr2**

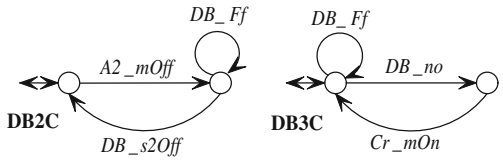


Fig. 29 Supervisors of **DB2**, **DB3**

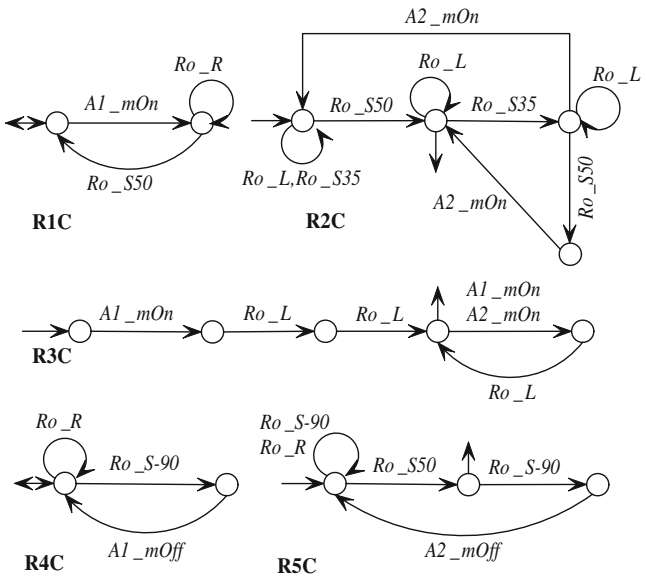


Fig. 30 Supervisors of **R1**, ..., **R5**

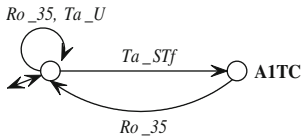


Fig. 31 Supervisor of **A1T**

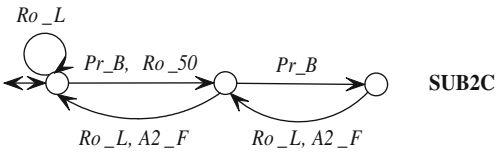


Fig. 32 Coordinator of subsystem 2

Table 8 Joint events

	SUB1	SUB2S
DB1C	*Cr_mOn, DB_Ff, DB_Sf, DB_s2Off, DB_no, DB_tau*	*A2_mOff*
DB2C	*DB_Ff, DB_s2Off*	*A2_mOff*
Ta2C	*Ta_D, Ta_STf*	*A1_mOn*
Ta4C	*Ta_L, Ta_S50f*	*A1_mOn*
A1TC	*Ta_U, Ta_STf*	*Ro_35*
	Cr_mOn, Ta_D, Ta_STf, Ta_U,	*A1_mOn*
	Ta_L, Ta_S50f, DB_Ff, DB_Sf,	*A2_mOff*
	DB_s2Off, DB_no, DB_tau	*Ro_35*

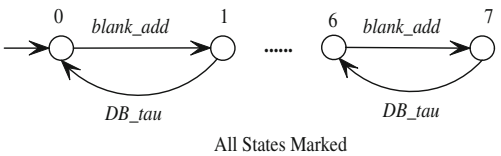


Fig. 33 Coordinator **TOTAL** of production cell

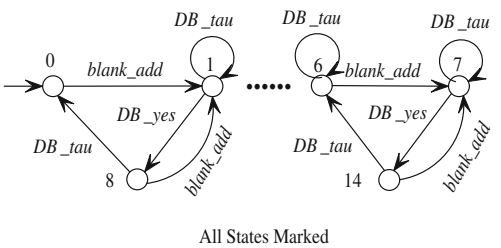


Fig. 34 Optimal coordinator **NTOTAL**

References

1. Ho YC (ed) (1992) Discrete event dynamic systems: analyzing complexity and performance in the modern world. IEEE Press, New York
2. Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event systems. SIAM J Control Optim 25(1):635–650
3. Ramadge PJ, Wonham WM (1989) The control of discrete event systems. Proc IEEE, Special Issue on DES 77(1):81–89
4. Brandin BA, Charbonnier FE (1994) The supervisory control of the automated manufacturing system of the AIP. In: Proceedings of the 4th International Conference on Computer Integrated Manufacturing and Automation Technology, New York, NY, pp 319–324. IEEE Computer Society Press
5. Chandra V, Huang Z, Kumar R (2003) Automated control synthesis for an assembly line using discrete-event system control theory. IEEE T Syst Man Cy C 33(2):284–289
6. Ricker S, Sarkar N, Rudie K (1996) A discrete-event system approach to modeling dexterous manipulation. Robotica 14 (5):515–526
7. Giua A, Seatzu C (2001) Supervisory control of railway networks with Petri nets. In: Proceedings of the 40th IEEE Conference on Decision and Control, Orlando, FL, December, 5004–5009
8. Jafari MA, Darabi H, Boucher TO, Amini A (2002) A distributed discrete event dynamic model for supply chain of business enterprises. In: Silva M, Giua A, Colom JM (eds) Proceedings of WODES2002, Zaragoza, Spain, pp 279–285. IEEE Control Systems Society
9. Kozak P, Wonham WM (1996) Design of transaction management protocols. IEEE T Automat Contr 41(9):1330–1335
10. Feng L, Wonham WM, Thiagarajan PS (2007) Designing communicating transaction processes by supervisory control theory. Form Method Syst Des 30(2):117–141
11. Seidl M (2006) Systematic controller design to drive high-load cell centers. IEEE T Contr Syst T 14(2):216–223
12. de Queiroz MH, Cury JER (2000) Modular supervisory control of large scale discrete event systems. In: Boel R, Stremersch G (ed) Discrete event systems: analysis and control. Kluwer, Dordrecht, pp 103–110
13. Hoffman G, Wong-Toi H (1992) Symbolic synthesis of supervisory controllers. In: Proceedings of ACC, Chicago, IL, pp 2789–2793
14. Åkesson K, Flordal H, Fabian M (2002) Exploiting modularity for synthesis and verification of supervisors. In: Proceedings of the 15th IFAC World Congress on Automatic Control, Barcelona, Spain
15. Ma C, Wonham WM (2005) Nonblocking supervisory control of state tree structures. In: Thoma M, Morari M (ed) LNCIS 317. Springer, Berlin Heidelberg New York
16. Schmidt K, Marchand H, Gaudin B (2006) Modular and decentralized supervisory control of concurrent discrete event systems using reduced system models. In: Lafortune S, Lin F, Tilbury D (ed) Proceedings of the 8th WODES, Ann Arbor, MI, July, pp 149–154
17. Zhang ZH (2001) Smart TCT: an efficient algorithm for supervisory control design. M.A.Sc thesis, Electrical and Computer Engineering, University of Toronto
18. Hill R, Tilbury D (2006) Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In: Lafortune S, Lin F, Tilbury D (ed) Proceedings of the 8th WODES, Ann Arbor, MI, July, pp 399–406
19. Gohari P, Wonham WM (2000) On the complexity of supervisory control design in the RW framework. IEEE T Syst Man Cy B 30 (5):643–652
20. Rohloff K, Lafortune S (2002) On the computational complexity of the verification of modular discrete-event systems. In: Proceedings of the 41th CDC, Las Vegas, NV, December, 16–21
21. Lee SH, Wong KC (2002) Structural decentralized control of concurrent discrete-event systems. Eur J Control 8(5):477–491
22. Schmidt K, Reger J, Moor T (2004) Hierarchical control for structural decentralized DES. In: Proceedings of the 7th WODES, Rheims, France, September, pp 289–294
23. Wong KC, Wonham WM (1998) Modular control and coordination of discrete-event systems. Discrete Event Dyn Syst 8(3):247–297
24. Leduc RJ, Brandin BA, Lawford M, Wonham WM (2005) Hierarchical interface-based supervisory control-part I: serial case. IEEE T Automat Contr 50(9):1322–1335
25. Leduc RJ, Lawford M, Wonham WM (2005) Hierarchical interface-based supervisory control-part II: parallel case. IEEE T Automat Contr 50(9):1336–1348
26. Pena P, Cury J, Lafortune S (2006) Testing modularity of local supervisors: an approach based on abstractions. In: Lafortune S, Lin F, Tilbury D (ed) Proceedings of the 8th WODES, Ann Arbor, MI, July, pp 107–112
27. Wong KC, Wonham WM (1996) Hierarchical control of discrete-event systems. Discrete Event Dyn Syst 6(3):241–273
28. Vahidi A, Fabian M, Lennartson B (2006) Efficient supervisory synthesis of large systems. Control Eng Pract 14(10):1157–1167
29. Song R, Leduc RJ (2006) Symbolic synthesis and verification of hierarchical interface-based supervisory control. In: Lafortune S, Lin F, Tilbury D (ed) Proceedings of the 8th WODES, Ann Arbor, MI, July, pp 419–426
30. Feng L, Wonham WM (2006) Computationally efficient supervisor design: abstraction and modularity. In: Lafortune S, Lin F, Tilbury D (ed) Proceedings of the 8th WODES, Ann Arbor, MI, July, 3–8
31. Feng L, Wonham WM (2006) Computationally efficient supervisor design: control flow decomposition. In: Lafortune S, Lin F, Tilbury D (ed) Proceedings of the 8th WODES, Ann Arbor, MI, July, 9–14
32. Feng L, Wonham WM (2007) Nonblocking coordination of discrete-event systems by control-flow nets. In: Proceedings of the 46th CDC, New Orleans, LA, December
33. Feng L (2007) Computationally efficient supervisory design for discrete-event systems, PhD Thesis, Electrical and Computer Engineering, University of Toronto, 2007, http://www.control.utoronto.ca/~fenglei
34. Feng L, Wonham WM (2008) Supervisory control architecture for discrete-event systems. IEEE T Automat Contr (to appear in June 2008)
35. Holloway LE, Krogh BH, Giua A (1997) A survey of Petri net methods for controlled discrete event systems. Discrete Event Dyn Syst 7(2):151–190
36. Iordache MV, Antsaklis PJ (2006) Supervision based on place invariants: a survey. Discrete Event Dyn Syst 16(4):419–557
37. Giua A, DiCesare F, Silva M (1992) Generalized mutual exclusion constraints on nets with uncontrollable transitions. In: Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Chicago, IL, pp 974–979
38. Uzam M, Wonham WM (2006) A hybrid approach to supervisory control of discrete event systems coupling RW supervisors to Petri nets. Int J Adv Manuf Technol 28(7):747–760

39. Fanti MP, Zhou MC (2004) Deadlock control methods in automated manufacturing systems. IEEE T Syst Man Cy A 34(1):5–22
40. Zajac J (2004) A deadlock handling method for automated manufacturing systems. Ann CIRP Manuf Tech 53(1):367–370
41. Wonham WM (2007) Supervisory control of discrete-event systems. ECE Dept, University of Toronto, 2002–2007, http://www.control.toronto.edu/DES, Updated 2007.07.01
42. Cassandras CG, Lafortune S (1999) Introduction to discrete event systems. Kluwer, Dordrecht
43. Feng L, Wonham WM (2006) On the computation of natural observers in discrete-event systems. In: Proceedings of the 45th IEEE CDC, San Diego, CA, December, pp 428–433
44. Wong KC, Wonham WM (2004) On the computation of observers in discrete-event systems. Discrete Event Dyn Syst 14(1):55–107
45. XPTCT (2007) Design software for SCT (Version 119, Windows XP, updated 2007.07.01). ECE Dept, University of Toronto, http://www.control.toronto.edu/DES
46. Melcher H, Winkelmann K (1998) Controller synthesis for the 'production cell' case study. Proceedings of the second workshop on formal methods in software practice, Clearwater Beach, FL, pp 24–33
47. Su R, Wonham WM (2004) Supervisor reduction for discrete-event systems. Discrete Event Dyn Syst 14(1):31–53