Int J Adv Manuf Technol (2010) 50:1189–1202
DOI 10.1007/s00170-010-2561-2

ORIGINAL ARTICLE

Supervisor localization for large discrete-event systems

Case study production cell

Kai Cai · W. M. Wonham

Received: 3 October 2009 / Accepted: 1 February 2010 / Published online: 7 March 2010

Abstract We study the design of distributed control for large-scale discrete-event systems. Our approach, called *supervisor localization*, allocates external supervisory control action to individual plant components as their internal control strategies. The approach is demonstrated in detail on a benchmark application, the Production Cell: a distributed control architecture is established wherein every component acquires a set of local controllers. Further, we provide a quantitative analysis of tradeoffs between the distributed and decentralized architectures, thereby pointing the way to criteria for architectural choice.

Keywords Discrete-event systems · Distributed supervisory control · Supervisor localization

1 Introduction

Rapid developments of embedded and network technologies have made low-cost computing power and efficient communication pervasive in engineering practice. These technological advances have brought about significant changes in the design and implementation of many large-scale industrial systems (like manufacturing cells and chemical plants). In particular, there is a trend toward systems of smart agents, where systemic computation is distributed among individual component agents, and global supervision is replaced with agents' local decision-making supported by information exchange among peers.

In the spirit of smart agent systems, we proposed [1–4] a novel *distributed control* paradigm for discrete-event systems (DES) in the framework of supervisory control theory (SCT) [13, 22]. We assume that the plant to be controlled comprises independent asynchronous components, which are coupled implicitly through control specifications.[1] The objective of distributed control is to allocate control action to each individual component in such a way that the resulting private or *localized* controllers collectively achieve optimal (i.e., minimally restrictive) and nonblocking controlled behavior for the overall system. Under this scheme, each private controller controls only its own (controllable) events, although it may very well need to observe events originating in other components.

Distinct, though related, control architectures are decentralized, hierarchical, and heterarchical (e.g., [6, 20, 24]). Both distributed and these modular approaches aim to achieve efficient computation and transparent control logic, while realizing global optimality and nonblockingness. A structural distinction, however, is that, with modular supervision, the global control action is typically allocated among specialized supervisors enforcing individual specifications. By contrast, with our distributed supervision, it is allocated

This work was supported in part by the Natural Sciences and Engineering Research Council (Canada), Grant no. 7399.

K. Cai (✉) · W. M. Wonham
Systems Control Group, Department of Electrical and Computer Engineering, University of Toronto, Toronto, M5S 3G4, Ontario, Canada
e-mail: caikai@control.utoronto.ca

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

[1] This assumption is commonly adopted in dealing with practical DES control problems (e.g., [6, 9, 19]).

among the individual active (i.e., plant) components. Intuitively, one could think of the modular architectures as employing 'intelligent' supervisors to direct 'dumb' components, while our distributed paradigm as equipping components with embedded controllers, thereby turning them into 'smart agents'.

To address control design in the distributed architecture, we developed [1–4] an algorithm called *supervisor localization* (SL), which decomposes a synthesized monolithic supervisor into local controllers for individual active components. It is then proved that the family of local controllers (one for each component) provides the same global control action as the monolithic supervisor did—and is therefore optimal and nonblocking for the entire plant. In the case of large-scale systems,[2] owing to state explosion the monolithic supervisor may not be feasibly computable. Indeed, Gohari and Wonham [8] proved that the monolithic supervisor synthesis is NP-hard, inasmuch as the state space size grows exponentially in the number of individual plant components and specifications. To manage such complexity, we proposed [3, 4] combining the SL algorithm with a flexible heterarchical architecture [6] which reduces computational effort in localization. This combination leads to a *decomposition-aggregation procedure* (DAP): First design an organization of modular supervisors that achieves global optimality and nonblockingness; then apply SL to decompose each of these modular supervisors into local controllers for the relevant components.

We note that a recent paper [16] proposed a scheme similar in general terms to our own, but proposed a control synthesis that amounts merely to making copies of the (reduced) monolithic supervisor for each component, with certain corresponding, extraneous selfloops removed. By contrast, our SL algorithm is suitably adapted from *supervisor reduction* [18]—for each component in turn, we reduce a given supervisor in a way that depends on the controllable events specific to that component alone—thus achieving a truly local result. Moreover, [16] did not provide an approach to large systems, while our proposed DAP [3, 4] is an effective solution procedure that combines the SL algorithm with a (more-or-less universal) modular architecture [6].

In this paper and its conference precursor [3], we study in detail the distributed control design of a benchmark application, the Production Cell, having state size of order 10^8. The primary contribution of this study is the demonstration that the SL algorithm [1–4] and the modular theory in [6] can be combined to provide an effective attack on large-scale systems. Further, in the present paper, we provide detailed tradeoffs between the localization result for the Production Cell and the decentralized result [7] previously derived for the same example. This type of analysis could help a designer to choose between the competing control architectures.

The rest of the paper is organized as follows: First, in Section 2, we briefly review the distributed control theory for large-scale DES. In Sections 3 and 4, we describe the Production Cell system and present the solution to the distributed control problem. Then, we make quantitative comparisons between the distributed and decentralized architectures in Section 5. Finally, we state conclusions in Section 6.

2 Distributed control theory

2.1 Problem formulation

The plant to be controlled is modelled by a (nonempty) *generator* $\mathbf{G}$ defined over a (finite) *alphabet* Σ, with *closed* and *marked languages* $L(\mathbf{G})$ and $L_m(\mathbf{G})$ [22]. The alphabet Σ is partitioned into the controllable event subset Σ_c and the uncontrollable subset Σ_u, written $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$. Consider the case where $\mathbf{G}$ consists of asynchronous components $\mathbf{G}^k$ ($k \in \mathcal{K}$, $\mathcal{K}$ an index set), called 'active'. Namely, the $\mathbf{G}^k$ are defined over pairwise disjoint alphabets Σ^k, with $\Sigma = \dot{\bigcup}\{\Sigma^k | k \in \mathcal{K}\}$. Let $L_k := L(\mathbf{G}^k)$ and $L_{m,k} := L_m(\mathbf{G}^k)$; then, we have $L(\mathbf{G}) = ||\{L_k | k \in \mathcal{K}\}$ and $L_m(\mathbf{G}) = ||\{L_{m,k} | k \in \mathcal{K}\}$, where "||" denotes *synchronous product* [22]. For simplicity, we assume that, for every $k \in \mathcal{K}$, $\mathbf{G}^k$ is *nonblocking* (i.e., the *prefix-closure* [22] $\bar{L}_{m,k} = L_k$). Then, $\mathbf{G}$ is necessarily nonblocking (i.e., $\bar{L}_m(\mathbf{G}) = L(\mathbf{G})$).

The components are implicitly coupled through a control specification language E that imposes behavioral constraints on $\mathbf{G}$. As in the literature (e.g., [12, 19]), assume that E is *decomposable* into specifications $E_p \subseteq \Sigma_{e,p}^*$ ($p \in \mathcal{P}$, $\mathcal{P}$ an index set), where the $\Sigma_{e,p} \subseteq \Sigma$ need not be pairwise disjoint; namely, $E = ||\{E_p | p \in \mathcal{P}\}$. Thus, E is defined over $\Sigma_e := \bigcup\{\Sigma_{e,p} | p \in \mathcal{P}\}$. Let $P_e : \Sigma^* \to \Sigma_e^*$ be the corresponding *natural projection* [22], and write $P_e^{-1} : \mathrm{Pwr}(\Sigma_e^*) \to \mathrm{Pwr}(\Sigma^*)$ for the inverse-image function of P_e, where $\mathrm{Pwr}(\cdot)$ denotes powerset.

Let $F \subseteq \Sigma^*$, and recall that F is *controllable* [22] (with respect to $\mathbf{G}$) if $\bar{F}\Sigma_u \cap L(\mathbf{G}) \subseteq \bar{F}$ (where $\bar{F}$ is the prefix-closure of F). Whether or not F is controllable,

[2] We take the pragmatic view that a system is large-scale whenever "it is made up of a large number of parts that interact in a nonsimple way" [17].

we denote by $\mathcal{C}(F)$ the family of all controllable sublanguages of F. Then, $\mathcal{C}(F)$ is nonempty and contains a (unique) supremal element, denoted $\sup\mathcal{C}(F)$ [22, 23].

For the plant **G** and the specification E described above, let a generator **SUP** (over Σ) be the corresponding monolithic supervisor that is optimal and nonblocking. Since we are concerned with large-scale DES, assume that **SUP** is not feasibly computable. The marked language of **SUP** can, nevertheless, be expressed algebraically as $L_m(\mathbf{SUP}) = \sup\mathcal{C}(P_e^{-1}E \cap L_m(\mathbf{G}))$.

Now we define *local controllers* for individual components. With $\Sigma = \Sigma_c \dot\cup \Sigma_u$, we assign control structure to each component as follows:

$$\Sigma_c^k = \Sigma^k \cap \Sigma_c,\ \ \Sigma_u^k = \Sigma^k \cap \Sigma_u.$$

Fix $k \in \mathcal{K}$. We say that a generator $\mathbf{LOC}^k$ (over Σ) is a *local controller* for component $\mathbf{G}^k$ if $\mathbf{LOC}^k$ can disable only events in Σ_c^k. Precisely, for all $s \in \Sigma^*$ and $\sigma \in \Sigma$, there holds

$$s\sigma \in L(\mathbf{G}),\ s \in L\big(\mathbf{LOC}^k\big),\ s\sigma \notin L\big(\mathbf{LOC}^k\big) \Rightarrow \sigma \in \Sigma_c^k.$$

As to its observation, in nontrivial cases[3] $\mathbf{LOC}^k$ observes, and responds to, events generated by components other than $\mathbf{G}^k$, thereby ensuring correct local control decisions. Thus, while a local controller's control authority is strictly local, its observation scope need not, and generally will not, be.[4]

We emphasize that, in supervisor localization, the observation scope of each local controller emerges as part of the result and is not specified a priori. How small that scope can be will depend on how close our specialized supervisor reduction algorithm (of polynomial complexity) is to being optimal (which is NP-hard); see [4] for further details. This flexibility considerably reduces the practical difficulties of implementing the localized results, as compared to approaches evolved from co-observability [14], where the directly observable event sets of decentralized controllers are laid down in advance (for a recent review, see, e.g., [10]).

We are ready to formulate the distributed control problem (⋇): Construct for each component $\mathbf{G}^k$ ($k \in \mathcal{K}$) a set of local controllers $\mathbf{LOC}^k = \{\mathbf{LOC}_{i_k}^k | i_k \in \mathcal{I}_k\}$ ($\mathcal{I}_k$ an index set), with $L(\mathbf{LOC}^k) = \bigcap\{L(\mathbf{LOC}_{i_k}^k) | i_k \in \mathcal{I}_k\}$ and $L_m(\mathbf{LOC}^k) = \bigcap\{L_m(\mathbf{LOC}_{i_k}^k) | i_k \in \mathcal{I}_k\}$. Further, let $\mathbf{LOC} = \{\mathbf{LOC}^k | k \in \mathcal{K}\}$ be the family of all local controllers, with $L(\mathbf{LOC}) = \bigcap\{L(\mathbf{LOC}^k) | k \in K\}$ and $L_m(\mathbf{LOC}) = \bigcap\{L_m(\mathbf{LOC}^k) | k \in K\}$. It is then required that

$$L(\mathbf{G}) \cap L(\mathbf{LOC}) = L(\mathbf{SUP}) \tag{1a}$$

$$L_m(\mathbf{G}) \cap L_m(\mathbf{LOC}) = L_m(\mathbf{SUP}) \tag{1b}$$

We say that the collection **LOC** of local controllers, satisfying (1a) and (1b), is *control equivalent* to **SUP** with respect to **G**.

For the sake of easy implementation and comprehensibility, it would be desired in practice that the state sizes of local controllers be very much less than that of their 'parent' monolithic supervisor; that is, $(\forall k \in \mathcal{K}, \forall i_k \in \mathcal{I}_k)\ |\mathbf{LOC}_{i_k}^k| \ll |\mathbf{SUP}|$, where $|\cdot|$ denotes state size of the argument. Inasmuch as this property is neither precise to state nor always achievable, it must needs be omitted from the formal problem statement; in applications, nevertheless, it should be kept in mind.

2.2 Solution procedure

We outline the solution procedure DAP to the distributed control problem (⋇) for large-scale DES. This procedure consists of seven steps, of which the first six systematically synthesize a group of modular supervisors that achieves global optimal and nonblocking control [6], and the last step applies the SL algorithm [1–4] to decompose each of those synthesized supervisors into local controllers. We now illustrate DAP through the simple but representative example displayed in Figs. 1 and 2; for a formal presentation of the procedure, we refer the reader to [3, 4].

1. *Plant model abstraction:* Part of the plant dynamics that is unrelated to the imposed specifications may be concealed. By hiding irrelevant transitions, we can simplify the models of components. The technique for model abstraction is to find for each component a natural projection satisfying *observer* and *output control consistent* properties [6, Sections II,

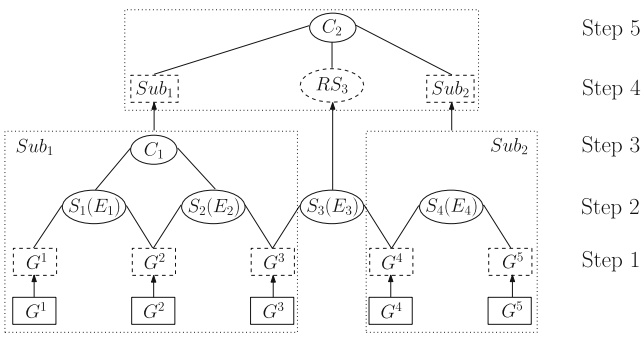


Fig. 1 Modular supervisor synthesis

[3] See [1, Section 2.6], [4, Section II-D], for the trivial case where no coupling among components is imposed by specifications.

[4] For simplicity, we assume in this paper that observation of an event is simultaneous with its occurrence.

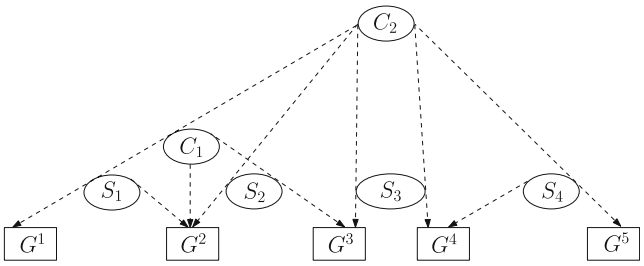


Fig. 2 Modular supervisor localization

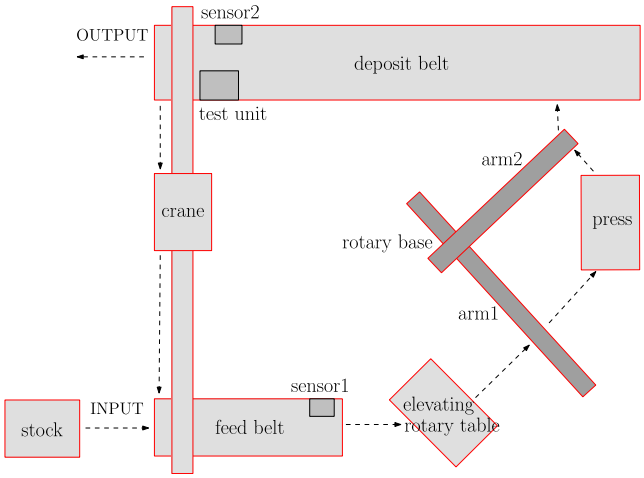


Fig. 3 Top view of production cell

III]; such a projection preserves critical information about the original component models with respect to optimality and nonblockingness. For illustration, see "Step 1" in Fig. 1, where the component G^i ($i = 1, ..., 5$) with a dashed box denotes its abstraction.

2. *Decentralized supervisor synthesis:* The system now consists of component model abstractions and control specifications. Since each specification may impose coupling only on a subset of component abstractions, a corresponding optimal and nonblocking decentralized supervisor can be synthesized, by standard methods [22, 23], based only on those relevant abstractions. This is displayed in Fig. 1, "Step 2," where E_i ($i = 1, ..., 4$) denotes a specification and S_i the corresponding decentralized supervisor.
3. *Subsystem decomposition and coordination:* After synthesizing decentralized supervisors, we view the whole system as comprised of a set of modules, each consisting of a decentralized supervisor with associated component abstractions. In this step, we decompose the overall system into small-scale subsystems, through grouping these modules based on their interconnection dependencies (e.g., event-coupling). If the modules admit certain special structure, *control-flow net* [5] is an effective approach for subsystem decomposition.
   Having obtained a group of small subsystems, we verify the nonblocking property for each of them.[5] If a subsystem happens to be blocking, we design a *coordinator*[6] to resolve the conflict [6, Proposition 7, Theorem 4]. For the example in Fig. 1, in "Step 3," we decompose the system consisting of four modules into two subsystems (Sub_1 and Sub_2), leaving the supervisor S_3 in between. In case Sub_1 is blocking (i.e., the two supervisors S_1 and S_2 are conflicting), a coordinator C_1 is designed to resolve this conflict.
4. *Subsystem model abstraction:* After ensuring nonblockingness within each subsystem, we need to verify the nonconflicting property for the group of subsystems. Directly verifying this property requires expensive computation; instead, we again apply the technique of model abstraction to simplify every subsystem, followed by the nonconflicting check at the abstracted level. This is illustrated in Fig. 1, "Step 4," where the subsystem Sub_i ($i = 1, 2$) with a dashed box denotes its abstraction. In addition, for the intermediate supervisor S_3, we apply the reduction algorithm [18] to obtain its (control-equivalent) reduced model, denoted by RS_3.
5. *Abstracted subsystem decomposition and coordination:* This step is analogous to Step 3), but for subsystem model abstractions instead of modules. Concretely, we organize subsystem abstractions into groups according to their interconnection dependencies (e.g., event-coupling). Again, control-flow net may be an effective tool if a certain special structure is present. Then, for each group, we check if the included subsystem abstractions are nonconflicting, and if not, design a coordinator to

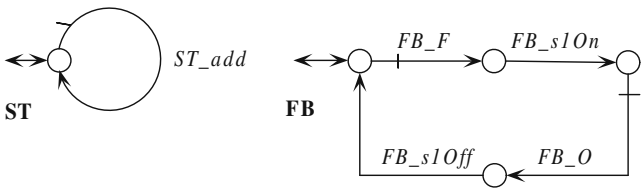


Fig. 4 Plant models of stock and feed belt

[5] We use TCT [21] procedure **nonconflict** for this verification.

[6] A coordinator is a generator that does not directly enforce a 'safety' specification, but only resolves conflict among decentralized supervisors. In other words, a coordinator enforces only a nonblocking specification.

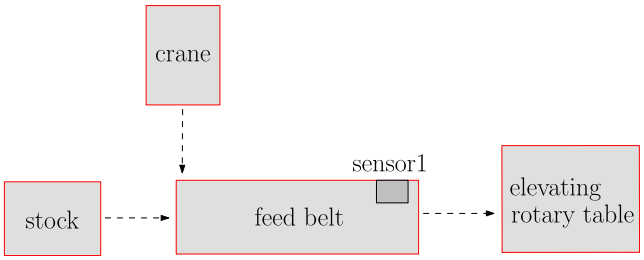


Fig. 5 Interconnection of feed belt

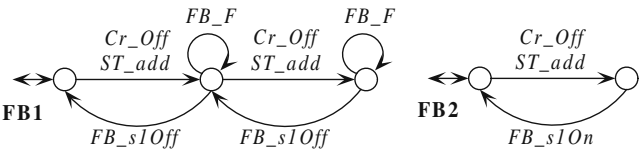


Fig. 6 Specification models of feed belt

resolve the conflict. In Fig. 1, "Step 5," we treat the two subsystem abstractions and the intermediate reduced supervisor as a single group. If this group turns out to be blocking, another coordinator C_2 is designed to resolve the conflict.

6. *Higher-level abstraction:* Repeat Steps 4 and 5 until there remains a single group of subsystem abstractions in Step 5.
   The modular control design terminates at Step 6; we have obtained a hierarchy of decentralized supervisors and coordinators. Specifically, Step 2 produces a set of decentralized supervisors, and Steps 3–6 iteratively generate a set of coordinators. For the example in Fig. 1, a hierarchy of four decentralized supervisors and two coordinators has been synthesized.
   While, in the previous Steps 1–6, we have generally followed the prescription of [3, 4], other systematic approaches to the construction of heterarchical controls (e.g., [15]) could serve as well. In this sense, the required preparation is nonspecific for our localization Step 7, which follows.
7. *Decentralized supervisors and coordinators localization:* In this last step, we apply the SL algorithm to localize each of these decentralized supervisors and coordinators to local controllers for their relevant components. To determine if a component is related to a supervisor or coordinator, we employ the criterion called *control coupling*[7] [3, 4]. This step for the running example is displayed in Fig. 2, with dashed lines denoting the control coupling relations among supervisors/coordinators and components. Thus, we apply the localization algorithm only with respect to entities joined by dashed lines.

Now, we state the main result, proved in [4].

[7]The control coupling relation can be determined by inspecting the control data table generated by the TCT procedure **condat** [21].

Theorem 1 *The procedure DAP solves the distributed control problem* (⁎).

3 Production cell: system description

This section and the next apply DAP to solve the distributed control problem for a benchmark application, the Production Cell originating with [11], in a version we adapt from [7]. We suitably modify the modeling in [7] for clearer interpretation of the cell's physical operations; the order 10^8 of the system state size is kept the same. In [7], optimal and nonblocking decentralized supervision has been established by the modular approach in [6]. This result will now be carried further to achieve our objective of distributed control: that is, the allocation of the global control action among individual active components.

Production Cell consists of nine asynchronous components: stock, feed belt, elevating rotary table, rotary base, arm1, arm2, press, deposit belt, and crane. The cell processes workpieces, called "blanks," as displayed in Fig. 3. In the following, we describe the operation of each component.

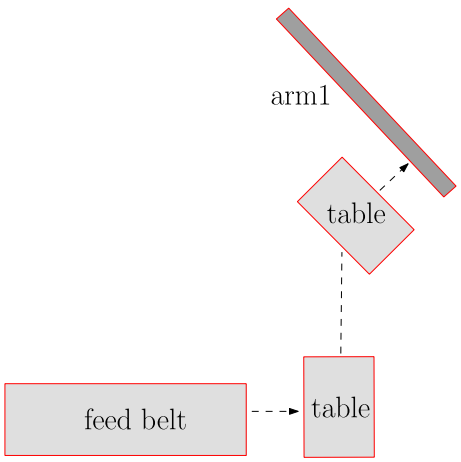


Fig. 7 Interconnection of elevating rotary table

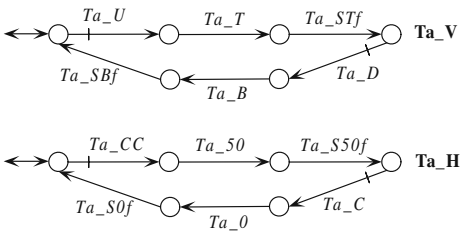


Fig. 8 Plant model of elevating rotary table

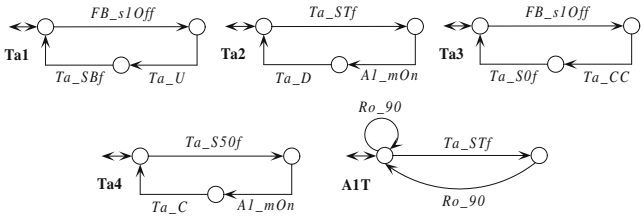


Fig. 9 Specification models of elevating rotary table

3.1 Stock

Stock adds blanks into the cell by placing them on feed belt (ST_add); see the generator **ST** in Fig. 4.[8]

3.2 Feed belt

Feed belt transports blanks towards table. According to FB in Fig. 4, once loaded, the feed belt forwards blanks (FB_F). Sensor1 at the end of the belt switches to "on" when it detects the arrival of a blank (FB_$s1On$). Feed belt outputs a blank onto table if the latter is available (FB_O). Sensor1 switches to "off" when a blank leaves (FB_$s1Off$).

Feed belt interacts with stock, crane, and table (as displayed in Fig. 5), subject to the following two specifications (Fig. 6):

- **FB1**: Feed belt forwards (FB_F) only when there are blanks loaded (by events Cr_Off or ST_add); and it can hold at most two blanks.
- **FB2**: If there is already one blank on feed belt, then for safety reasons a new blank is prohibited from being loaded before the first reaches the end of the belt and activates Sensor1 (FB_$s1On$).

3.3 Elevating rotary table

Table elevates and rotates in order to transfer blanks from feed belt to arm1; see the illustration in Fig. 7. As displayed in Fig. 8, the generators **Ta_V** and **Ta_H** describe table's vertical elevation and horizontal rotation, respectively. Thus, the complete behavior of table is represented by the synchronous product **Ta** := **Ta_V** || **Ta_H**. Specifically, after being loaded by feed belt, table moves up (Ta_U, Ta_T, Ta_STf) and turns counterclockwise (CCW) to $-50°$ (Ta_CC, Ta_50, Ta_$S50f$) for arm1 to pick up a blank. Thereafter, table moves down (Ta_D, Ta_B, Ta_SBf) and turns clockwise (CW) back to 0° (Ta_C, Ta_0,Ta_$S0f$).

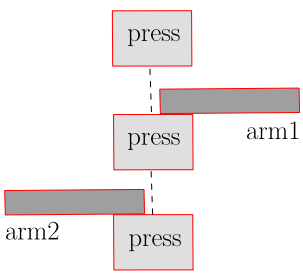
Table must synchronize with feed belt and arm1 when transferring blanks; the corresponding specifications are shown in Fig. 9:

- **Ta1** and **Ta3**: Table accepts a blank from feed belt (FB_$s1Off$) only when it is at bottom (Ta_SBf) and at angle 0° (Ta_$S0f$); only after accepting a blank is table permitted to move up (Ta_U) and turn CCW (Ta_CC).
- **Ta2** and **Ta4**: Table transports a blank to arm1 ($A1$_On) only when it is at top (Ta_STf) and at angle $-50°$ (Ta_$S50f$); only after transferring a blank is table permitted to move down (Ta_D) and turn CW (Ta_C).
- Collision between two blanks could occur if and when arm1 has been loaded with one blank by table, rotary base has not yet turned CCW to 90° (Ro_90), and table returns to top with a new blank (Ta_STf). The specification that prevents this collision is enforced by **A1T**, according to which table is not allowed to return to top before the loaded arm1 turns away to 90°.

3.4 Press

Press operates at three different positions: bottom, middle, and top (Fig. 10). According to Fig. 11, it is initially at bottom, and ascends to middle where arm1 may load a blank (Pr_UB, Pr_MU, Pr_SMf). After being loaded, press continues to top where it forges the blank (Pr_UM, Pr_T, Pr_STf). Then, it descends back

Fig. 10 Interconnection of press



[8] In generator models, by convention, we mark controllable events with a tick on the corresponding arrows.

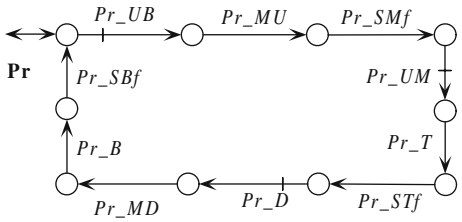


Fig. 11 Plant model of press

to bottom and prepares to unload the forged blank to arm2 (Pr_D, Pr_MD, Pr_B, Pr_SBf).

Press coordinates with arm1, arm2 as specified in Fig. 12:

- **Pr1**: Press can accept a blank from arm1 ($A1\_Off$) only at its middle position (Pr_SMf); only after accepting a blank can press move to top (Pr_UM).
- **Pr2**: Press can transfer a blank to arm2 ($A2\_On$) only at its bottom position (Pr_SBf); only after the transfer can press move to middle (Pr_UB).
- There are, additionally, two collision scenarios. First, press collides with arm1 if and when it is at top, arm1 is longer than 37, and rotary base is at 90°. Second, press collides with arm2 if and when it is not at bottom, arm2 is longer than 0, and rotary base is at 40°. The specifications for avoiding these collisions are enforced by **A1P** and **A2P**, according to which the respective three conditions in each case are prevented from being met simultaneously. Note that the initial and marked state of **A1P** is the state in the middle; this is because arm1 initially has length 52 (see **A1** in Fig. 14), and hence, the events Ro_90 and Pr_T cannot both occur before arm1 retracts to safe length ($A1\_37$). Similarly, the initial and marked state of **A2P** is also the middle one since rotary base is initially at 40° (refer to **Ro** in Fig. 14); consequently, the events $A2\_80$ and Pr_MU may not both happen before base turns to 90° (Ro_90).

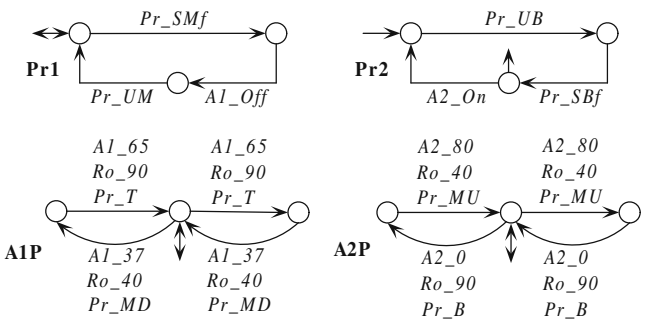


Fig. 12 Specification models of press

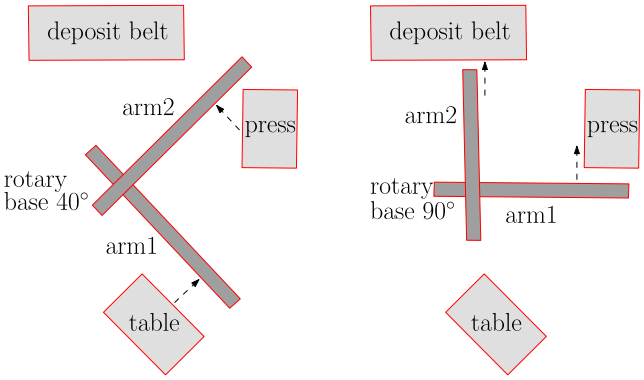


Fig. 13 Interconnection of rotary base, arm1, and arm2

3.5 Rotary base, arm1, and arm2

Rotary base, arm1, and arm2 cooperatively transfer blanks from table through press to deposit belt (see Fig. 13). As displayed in Fig. 14, rotary base initially at 40° rotates CCW to 90° (Ro_CC, Ro_90, Ro_$S90$), and then CW back to 40° (Ro_C, Ro_40, Ro_S40). Arm1, once loaded ($A1\_On$), first retracts to length 37 ($A1\_B37$, $A1\_37$, $A1\_S37$) so as to avoid collision, and then extends to length 65 ($A1\_F65$, $A1\_65$, $A1\_S65$), at which point it can unload a blank onto press ($A1\_Off$); after unloading arm1 retracts to its initial length 52 ($A1\_B52$, $A1\_52$, $A1\_S52$). Lastly, arm2 first extends its length to 80 ($A2\_F80$, $A2\_80$, $A2\_S80$), at which point it can pick up a blank from press ($A2\_On$); it then retracts to 57 ($A2\_B57$, $A2\_57$, $A2\_S57$) and places a blank onto deposit belt ($A2\_Off$); thereafter, it retracts to 0, its initial length ($A2\_B0$, $A2\_0$, $A2\_S0$).

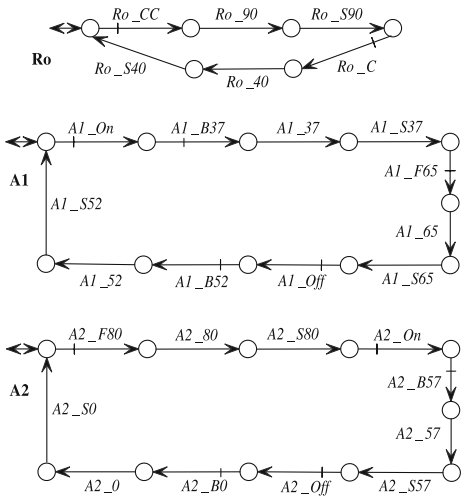


Fig. 14 Plant models of rotary base, arm1, and arm2

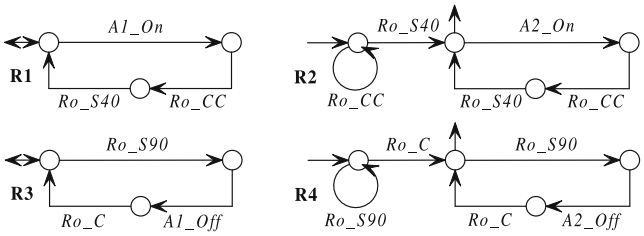


Fig. 15 Specification models of rotary base, arm1, and arm2

The collaboration among base and two arms must satisfy the specifications in Fig. 15:

- **R1** and **R2**: Arm1 and arm2 may be loaded only when base is at 40° (Ro_*S*40); only after both are loaded may base turn CCW (Ro_*CC*).
- **R3** and **R4**: Arm1 and arm2 may unload only when base is at 90° (Ro_*S*90); only after both unloading actions are completed may base turn CW (Ro_*C*).

Notice that the marked states in **R2** and **R4** are so chosen because arm2 has no blank to be loaded or to load during the first work cycle of base. An analogous reason accounts for the choice of marked state of **Pr2** in Fig. 12: Press has no blank to load arm2 for the first iteration of its actions.

3.6 Deposit belt

As shown in Fig. 16, once loaded, deposit belt forwards blanks (DB_*F*) towards the other end; there, Sensor2 switches to "on" (DB_*s*2*On*) when it detects the arrival of a blank, and "off" (DB_*s*2*Off*) to show that the blank has been checked by the test unit. If the blank passes the check (DB_*y*), then it will be output from the system (FB_*O*); otherwise (DB_*n*), it waits to be picked up by crane.

Deposit belt interacts with arm2 and crane (as displayed in Fig. 17), subject to the following three specifications (Fig. 18):

- **DB1**: Deposit belt forwards (DB_*F*) only when there are blanks loaded, and it can hold at most two blanks.

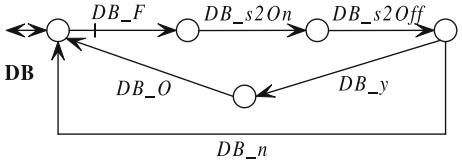


Fig. 16 Plant model of deposit belt

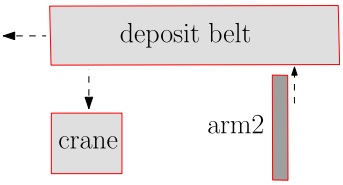


Fig. 17 Interconnection of deposit belt

- **DB2**: If there is already one blank on deposit belt, then, for safety, a new blank can be loaded only after the first is checked by the test unit (DB_*s*2*Off*).
- **DB3**: If a blank fails the test (DB_*n*), then it has to be taken by crane back for another cycle (Cr_*On*).

3.7 Crane

Crane transports faulty blanks from deposit belt to feed belt; see the illustration in Fig. 19. As shown in Fig. 20, the generators **Cr_V** and **Cr_H** describe, respectively, the vertical and horizontal motions of crane. Thus, the complete behavior of crane is represented by the synchronous product **Cr** := **Cr_V** || **Cr_H**. Concretely, after picking up a faulty blank from deposit belt (Cr_On), crane moves up (Cr_*U*, Cr_66, Cr_*SVf*) and horizontally towards feed belt (Cr_2*FB*, Cr_FB, Cr_*SHf*), to which it delivers the blank (Cr_Off). Thereafter, crane moves down (Cr_*D*, Cr_95, Cr_*SVf*) and horizontally back towards deposit belt (Cr_2*DB*, Cr_*DB*, Cr_*SHf*).

4 Production cell: distributed control

We are ready to apply the procedure **DAP** to the distributed control design for Production Cell.

1. *Plant model abstraction*: Projecting out the transitions that are unrelated to the imposed specifications, we effectively simplify the models of table, press, arm1, arm2, and crane, as displayed in Fig. 21.
2. *Decentralized supervisor synthesis*: For each specification, we group together its event-coupled component abstractions, and synthesize for each group

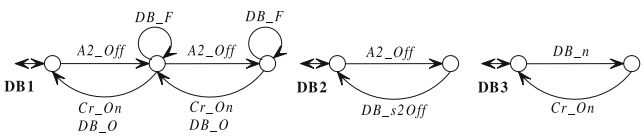


Fig. 18 Specification model of deposit belt

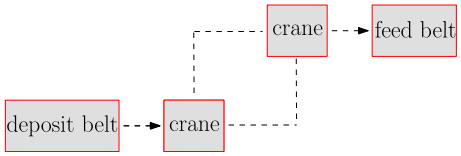


Fig. 19 Interconnection of crane

an optimal nonblocking decentralized supervisor using TCT procedure **supcon** [21]. The result is presented in Fig. 22, with component abstractions in blocks and supervisors in ovals, and solid lines denoting event-coupling. To display the generator models of these supervisors, we further apply TCT procedure **supreduce** [21]; the resulting reduced supervisors are shown in Fig. 23 (for clarity, extraneous selfloops are omitted).

3. *Subsystem decomposition and coordination*: We have obtained 18 decentralized supervisors; thus, we view the whole system as comprised of 18 modules, each consisting of a decentralized supervisor with associated component abstractions. Following [7], we decompose the overall system into two subsystems, leaving five supervisors in between, as shown with dotted lines in Fig. 22. It is further checked that Sub1 has 1,524 states and is nonblocking, while Sub2 has 535 states but turns out to be blocking. To resolve the conflict, we design for Sub2 a coordinator CO1, as displayed in Fig. 24; this coordinator forces arm2 to stay put at its initial state during the first work cycle of press. The resulting nonblocking subsystem is denoted by **NSub2**, having 339 states.
4. *Subsystem model abstraction*: We must now verify the nonconflicting property among Sub1, NSub2, and the five intermediate supervisors. For this, we again employ the abstraction technique to simplify the two subsystem models; the resulting abstractions are denoted by **Sub1**′ and **Sub2**′, having, respectively, 460 and nine states. Compared

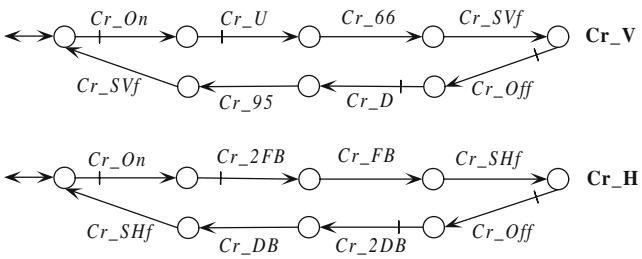


Fig. 20 Plant model of crane

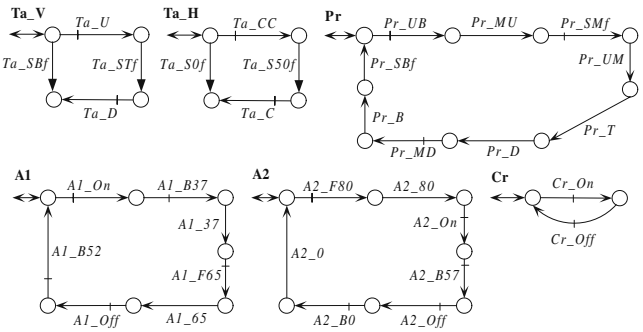


Fig. 21 Model abstractions of plant components

to their original state sizes, the two abstractions are economical. In addition, for the five intermediate supervisors, we take their (control-equivalent) reduced generator models (i.e., the results of the **supreduce** procedure in TCT [21]).

5. *Abstracted subsystem decomposition and coordination*: We treat **Sub1**′, **Sub2**′, and the five intermediate (reduced) supervisors as a single group, and directly check the nonblocking property. This group turns out, however, to be blocking, for the following reason. The whole cell contains a loop with material feedback, of faulty blanks, and the feedback event DB_*n* is uncontrollable. Thus, at least one empty slot must be maintained in the cell, because otherwise, if DB_*n* occurs, the loop will be 'choked'. Also, it can be checked that the total capacity of the cell is eight. Therefore, a coordinator **CO2** (see Fig. 24) is designed, which disables adding blanks (ST_add) into the cell if and when there are seven there already.
6. *Higher-level abstraction*: The modular supervisory control design terminates with the previous Step 5. We have obtained a hierarchy of 18 decentralized

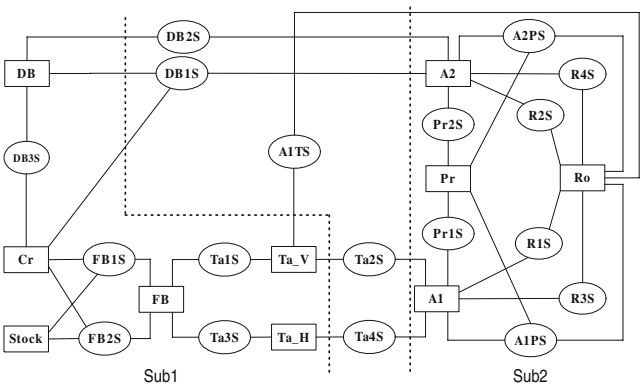


Fig. 22 Interconnection structure of production cell

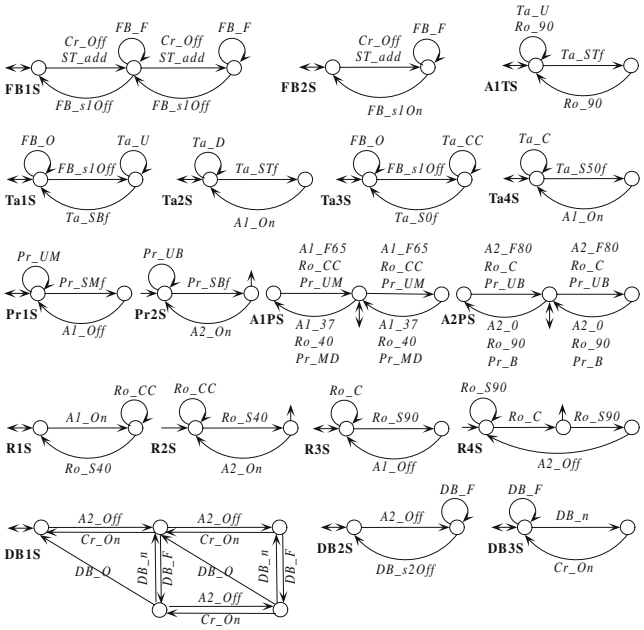


Fig. 23 Reduced generator models of decentralized supervisors

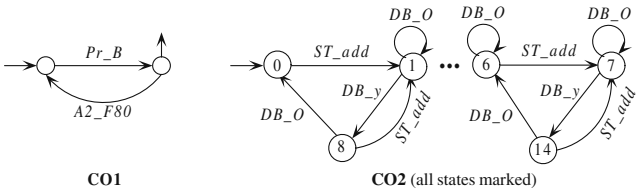


Fig. 24 Generator models of coordinators

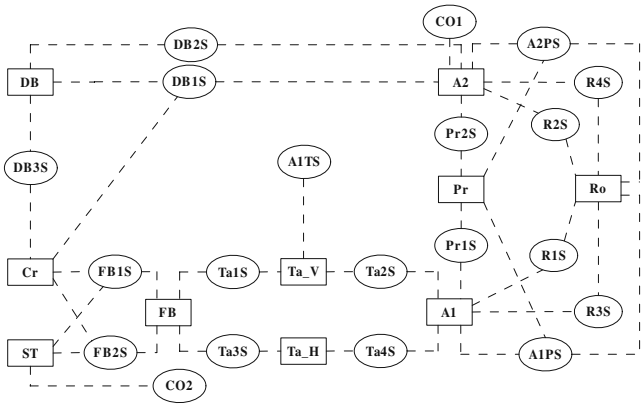


Fig. 25 Control-coupling relations

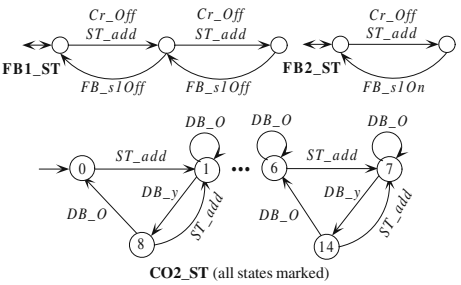


Fig. 26 Local controllers for stock

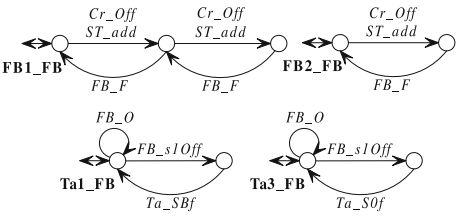


Fig. 27 Local controllers for feed belt

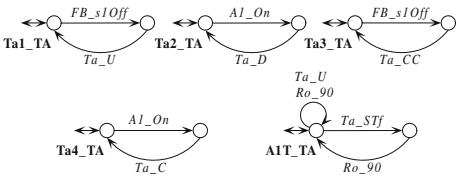


Fig. 28 Local controllers for elevating rotary table

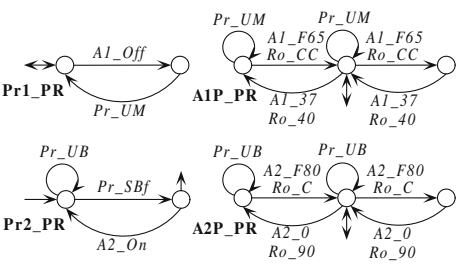


Fig. 29 Local controllers for press

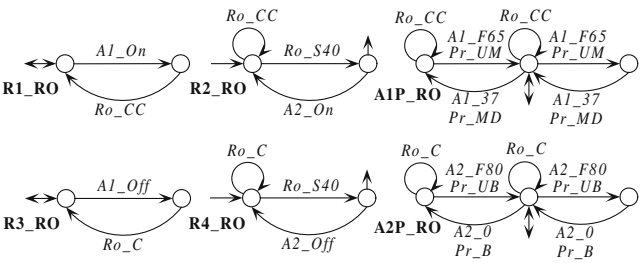


Fig. 30 Local controllers for robot

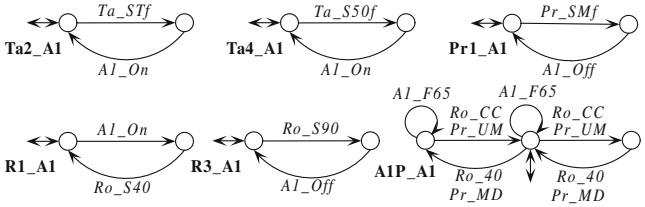


Fig. 31 Local controllers for arm1

supervisors and two coordinators; their synchronized behavior is globally optimal and nonblocking [7].

7. *Decentralized supervisors and coordinators localization*: We first determine the control-coupling relations between supervisors/coordinators and components; we do this by inspecting the corresponding **condat** tables in TCT [21]. The result is displayed in Fig. 25, with dashed lines denoting the control-couplings. For instance, **DB1S** disables (controllable) events in **DB**, **CR**, and **A2**. Then, we apply the **SL** algorithm to localize each supervisor/coordinator for its control-coupled component; the resulting local controllers are displayed in Figs. 26, 27, 28, 29, 30, 31, 32, 33, and 34 (for clarity, extraneous selfloops are omitted), grouped with respect to individual components. We observe that all controllers decomposed from the supervisors have only two to four states; thus, their control logic can be easily understood by referring to the corresponding specification descriptions in Section 3. Finally, by Theorem 1, these local controllers will collectively achieve global optimality and nonblocking.

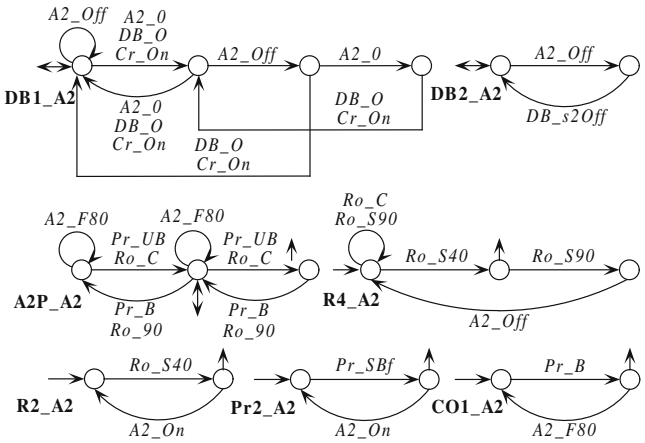


Fig. 32 Local controllers for arm2

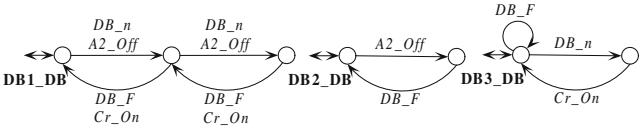


Fig. 33 Local controllers for deposit belt

5 Architectural comparisons

We have now presented the supervisor localization result for the Production Cell example. The result is a distributed control architecture wherein every component acquires a set of private controllers. In the present section, we compare this result with the decentralized result [7] previously derived for the same example. This investigation is important: having obtained these two distinct architectures, one naturally explores cost–benefit tradeoffs that may qualify one or the other as better suited to implement the tasks of the Production Cell. As the system modeling in [7] has been slightly modified, in Section 4, we therefore compare the result of Step 7 and that of Steps 1–6.

In general, comparing architectures quantitatively could involve many factors (such as state size, computing load, and sensing/communication scope), some of them not intrinsic to the architecture itself, but all of them involving costs which will be case-dependent. Hence, we compare the distributed and decentralized architectures for each specific task (i.e., specification) in the Production Cell, taken individually; in each case, we analyze tradeoffs with respect to certain primary factors, thereby pointing the way to criteria for architectural choice.[9]

First, we present a task to which the distributed architecture may be better suited. Recall that, when a decentralized supervisor is localized, its control action is *always* distributed to the resulting local controllers, in the sense that each local controller can disable only the controllable events present in its associated plant component. Controlling fewer events than its parent supervisor, a local controller may be expected to have a smaller state space, a narrower observation scope, and simpler control logic. This is true in the following example.

Example 1 Consider the control specification **DB1** as displayed in Fig. 18, which imposes a behavioral

[9]We need not investigate the specification **A1T** and the two coordination tasks, because the corresponding supervisor and coordinators are control-coupled only to a single plant component, and hence, the decentralized and distributed controls are the same.

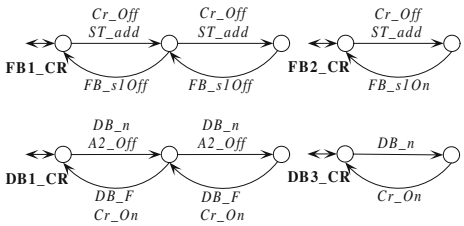


Fig. 34 Local controllers for crane

restriction on the components deposit belt, crane, and arm2. The restriction of **DB1** is equivalent to that of a buffer with capacity two, incremented by arm2, and decremented by deposit belt and crane. The objective is to protect the buffer against underflow and overflow, and, in addition, to prevent deposit belt forwarding (DB_*F*) if it is not loaded by arm2.

The corresponding decentralized supervisor, **DB1S**, is shown in Fig. 35. It has five states, three events (DB_*F*, Cr_On,*A*2_Off) to control, and five events (DB_*F*,DB_*n*, DB_*O*, Cr_On, *A*2_Off) to observe. Compared to the specification model, the increase in state size is because **DB1S** has to distinguish the two distinct paths, through either DB_*O* or DB_*n*, of reaching the marked state in the generator **DB** of deposit belt (see Fig. 16). Only when DB_*O* occurs can the buffer be decremented.

Now we localize **DB1S** for deposit belt, crane, and arm2, respectively, the result being displayed in Fig. 35. The local controller **DB1_DB** for deposit belt controls a single event DB_*F*, and thus, its control logic involves the disablement of solely this event. To convey the logic, the generator **DB1_DB** needs only three states, and the number of events that have to be observed reduces to four. The local controller **DB1_Cr** for crane is the same as that for deposit belt, except that the single event to be controlled is Cr_On. Lastly, the local controller **DB1_A2** for arm2 controls the event $A2\_Off$, and four states are needed to express the corresponding disablement logic. As to the observation scope, while the events DB_*F* and DB_*n* are excluded, **DB1_A2** has to observe the new event *A*2_0 which signals the completion of a work cycle of arm2 (refer to **A2** in Fig. 21).

We have thus seen that, for this task, **DB1**, every local controller is simpler than its parent supervisor **DB1S** in terms of control logic, state size, and observation scope. In practice, this means that the computing and sensing loads on each local controller may be less than that on the supervisor, and therefore, the distributed architecture may be a better choice.

It is not always, however, that the localized controllers will be simpler than their parent supervisor in every respect. There could often be tradeoffs between some factors, as illustrated in the case below.

Example 2 Consider the specification **R4** as displayed in Fig. 15, which imposes two constraints on the behaviors of base and arm2: (1) arm2 may unload ($A2\_Off$) only when base is at 90° (Ro_*S*90), and (2) only after the unloading is base permitted to turn CW (Ro_*C*). The corresponding decentralized supervisor, **R4S** in Fig. 36, has three states, two events (Ro_*C*, *A*2_Off) to control, and three events (Ro_*C*, Ro_*S*90, *A*2_Off) to observe.

Localizing **R4S** for base and arm2, respectively, we display the result in Fig. 36. The local controller **R4_RO** of base is responsible only for the disablement of Ro_*C*,

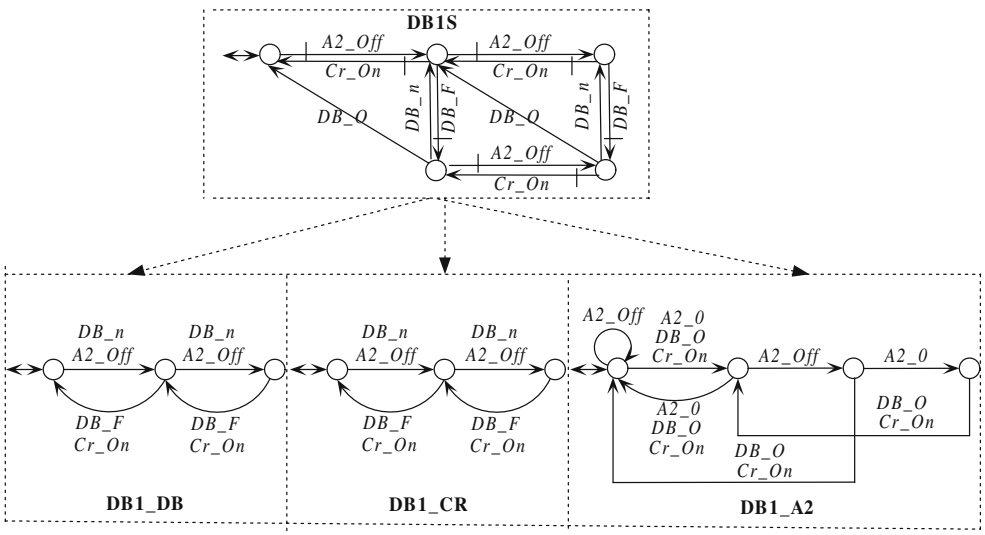


Fig. 35 Decentralized and distributed control for **DB1** task

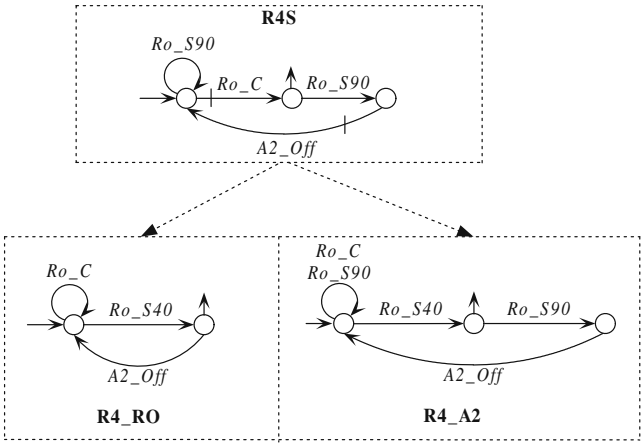


Fig. 36 Decentralized and distributed control for **R4** task

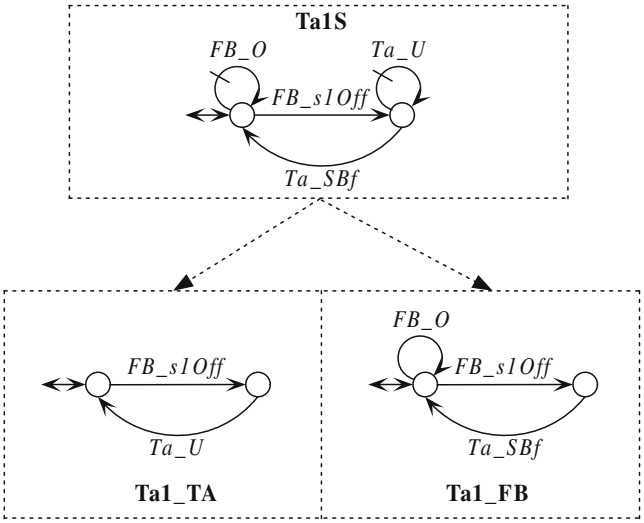


Fig. 37 Decentralized and distributed control for **Ta1** task

and its state size reduces to two. As to the observation scope, while the event Ro_S90 is excluded, **R4_RO** needs to observe the new event Ro_S40, which signals the completion of a work cycle of base (see **Ro** in Fig. 14). As to the local controller **R4_A2** of arm2, although it controls solely the event A2_Off, its state size remains at three, and even one more event Ro_S40 has to be observed compared to the supervisor **R4S**.

Thus, we have seen that, for the task **R4**, localization results in reduction in state size of one controller, but an increment in observation scope of the other controller. In practice, state size might relate to storage space and computing load, while observation scope could correspond to sensing or communication costs. It is left to the designer to weigh these factors according to the specific application at hand in order to choose one control architecture over the other.

Among the tasks in the Production Cell, the above two examples are the only cases where state space reduction is achieved. For all the remaining tasks, although each localized controller has the same number of states as the corresponding supervisor, it does enjoy a smaller observation scope, i.e., fewer events need to be observed. We exhibit one representative example below.

Example 3 Consider the specification **Ta1** as displayed in Fig. 9, which imposes two constraints on the behaviors of table and feed belt: (1) feed belt may unload (FB_$s1Off$) only when table is at bottom (Ta_SBf), and (2) only after the unloading may table ascend (Ta_U). The decentralized supervisor **Ta1S** enforcing this specification is shown in Fig. 37; it has two states, two events (Ta_U, FB_O) to control, and four events (Ta_U, Ta_SBf, FB_O, FB_$s1Off$) to observe.

The corresponding localized controllers for table and feed belt are displayed in Fig. 37. Again, each has two states; however, Ta1_TA does not observe the events Ta_SBf and FB_O, and Ta1_FB does not observe Ta_U. Thus, each local controller has a narrower observation scope compared to its parent supervisor. If, in an application, sensing or communication costs are of primary concern, the distributed architecture may be advantageous. On the other hand, if issues related to state size play a critical role, decentralized control may be preferred.

We have provided a preliminary analysis of detailed tradeoffs between the distributed and decentralized architectures involved in each particular task of the Production Cell. In the authors' view a more general formulation of cost-benefit architectural comparisons might turn out to have ample intrinsic interest, and awaits further development.

6 Conclusion

Applying the proposed decomposition-aggregation procedure [3, 4], we have successfully established a distributed control architecture for the benchmark Production Cell. In this architecture, every plant component is controlled by its own local controllers, while being coordinated with its fellows through their shared observable events. Such a control scheme enables distributed and embedded implementation of control action into individual components, which is the essence of the emergent smart agent systems.

In addition, we have compared the distributed architecture with a decentralized one which appeared in [7]. Concretely, we have analyzed detailed tradeoffs between the two architectures for each specific task in the Production Cell. Although the analysis is preliminary, it points to a general theory of control architecture, which we consider to be an ultimate objective of SCT.

References

1. Cai K (2008) Supervisor localization: a top-down approach to distributed control of discrete-event systems. Master's thesis, ECE Dept, University of Toronto. http://www.control.toronto.edu/DES/CaiKai_MASc_Thesis.pdf
2. Cai K, Wonham WM (2009a) Supervisor localization: a top-down approach to distributed control of discrete-event systems. In: Proc. 2nd Mediterranean conf. on intelligent systems and automation (CISA09), Zarzis, Tunisia, pp 302–308
3. Cai K, Wonham WM (2009b) Supervisor localization for large-scale discrete-event systems. In: Proc. 48th IEEE conf. on decision and control, Shanghai, pp 3099–3105
4. Cai K, Wonham WM (2010) Supervisor localization: a top-down approach to distributed control of discrete-event systems. IEEE Trans Automat Contr 55(3)
5. Feng L, Wonham WM (2006) Computationally efficient supervisory design: control flow decomposition. In: Proc. int. workshop discrete event systems (WODES06), Ann Arbor, pp 9–14
6. Feng L, Wonham WM (2008) Supervisory control architecture for discrete-event systems. IEEE Trans Automat Contr 53(6):1449–1461
7. Feng L, Cai K, Wonham WM (2009) A structural approach to the nonblocking supervisory control of discrete-event systems. Int J Adv Manuf Technol 41(11):1152–1167
8. Gohari P, Wonham WM (2000) On the complexity of supervisory control design in the RW framework. IEEE Trans Syst Man Cybern B Cybern 30(5):643–652 (Special Issue on DES)
9. Hill R, Tilbury D (2006) Modular supervisory control of discrete-event systems with abstraction and incremental hierarchical construction. In: Proc. int. workshop discrete event systems (WODES06), Ann Arbor, pp 399–406
10. Hiraishi K (2009) On solvability of a decentralized supervisory control problem with communication. IEEE Trans Automat Contr 54(3):468–480
11. Lewerentz C, Lindner T (eds) (1995) Formal development of reactive systems—case study production cell. Springer, London
12. Lin F, Wonham WM (1988) Decentralized supervisory control of discrete-event systems. Inf Sci 44:199–224
13. Ramadge PJ, Wonham WM (1987) Supervisory control of a class of discrete event processes. SIAM J Control Optim 25(1):206–230
14. Rudie K, Wonham WM (1992) Think globally, act locally: decentralized supervisory control. IEEE Trans Automat Contr 37(11):1692–1708
15. Schmidt K, Moor T, Perk S (2008) Nonblocking hierarchical control of decentralized discrete event systems. IEEE Trans Automat Contr 53(10):2252–2265
16. Seow KT, Pham MT, Ma C, Yokoo M (2009) Coordination planning: applying control synthesis methods for a class of distributed agents. IEEE Trans Control Syst Technol 17 (2):405–415
17. Simon HA (1962) The architecture of complexity. Proc Am Philos Soc 106:467–482
18. Su R, Wonham WM (2004) Supervisor reduction for discrete-event systems. Discrete Event Dyna Syst Theory Appl 14(1):31–53
19. Willner Y, Heymann M (1991) Supervisory control of concurrent discrete-event systems. Int J Control 54(5):1143–1169
20. Wong KC, Wonham WM (1996) Hierarchical control of discrete-event systems. Discrete Event Dyna Syst Theory Appl 6(3):241–273
21. Wonham WM (2009a) Design software: XPTCT, Systems Control Group, ECE Dept, University of Toronto, Version 133, Windows XP, updated July 1, 2009. http://www.control.toronto.edu/DES
22. Wonham WM (2009b) Supervisory control of discrete-event systems, Systems Control Group, ECE Dept, University of Toronto, updated July 1, 2009. http://www.control.toronto.edu/DES
23. Wonham WM, Ramadge PJ (1987) On the supremal controllable sublanguage of a given language. SIAM J Control Optim 25(3):637–659
24. Yoo TS, Lafortune S (2002) A general architecture for decentralized supervisory control of discrete-event systems. Discrete Event Dyna Syst Theory Appl 12(3):335–377