Int J Adv Manuf Technol (2007) 32: 1033–1044
DOI 10.1007/s00170-006-0421-x

ORIGINAL ARTICLE

Lin Li · Zhibin Jiang

A hybrid supervisory control approach for virtual production systems

Received: 13 June 2005 / Accepted: 28 November 2005 / Published online: 21 March 2006

Abstract With the development of computer technology, the anticipated extensive use of virtual production systems (VPSs) in the future has encouraged its intensive research recently. However, a limited amount of research has been made on its efficient control. In this paper, a hybrid supervisory control approach based on autonomy and coordination is proposed for VPSs to enhance the efficiency of control. Its three primary advantages are: (1) application of supervisory control theory, which can efficiently obtain the desired property through closed-loop feedback; (2) use of hybrid control structure combining hierarchical and distributed ones, which can avoid the exponential explosion of state space due to synthesis of models; and (3) utilization of autonomous and coordination mechanisms, which can keep the balance of local quick response and global optimization. A case study is used to illustrate how to implement the proposed approach, and is finally analyzed and simulated by an integrated tool named UPPAAL.

Keywords Autonomy and coordination · Hybrid supervisory control · Supervisory control theory · Virtual production systems (VPSs)

1 Introduction

To cope with today's customer-driven manufacturing (CDM) environment, Jiang et al. proposed a paradigm for organizing virtual production systems (VPSs) [1]. It combines the advantages of functional and product-oriented resources structures, such that both efficient internal logistics management and convenient production resources management are enabled. With the development of computer technology, the anticipated extensive use of VPSs in the future has encouraged its intensive research on modeling and scheduling methodologies in recent years, mostly based on Petri nets [2, 3]. However, a limited amount of research has been made on its efficient control. Since Petri-nets-based control model is a lack of supervisory function (feedback) in the process of control, it requires iteration through phases of trial, analysis and redesign to converge to the desired property [4]. For complex VPSs, this may result in a time-consuming calculation and slow down its response speed.

Compared with Petri nets, supervisory control theory proposed by Ramadge and Wonham [5] provides the modeling features similar to Petri nets, but it is more efficient in control than the latter. The reason is that it may ensure the system under control to behave legally according to given specifications by closed-loop feedback instead of iterating them. To enhance the ability of modeling, analysis and simulation, supervisory control theory has made more extended research. On one hand, new automata and control structures were continuously proposed to solve the explosion of state space due to synthesis of automata in complex discrete events systems (DES) [6–8]. And on the other hand, supervisory control theory is in conjunction with computer technologies. Notably, a number of verification tools [9–11] were developed, which have been successfully applied in industrial case studies. In these tools, UPPAAL tool [11] is an integrated environment for modeling, simulation and verification of real-time systems modeled as networks of timed automata, which was developed in collaboration between the Uppsala University in Sweden and the Aalborg University in Demark.

Motivated by the above consideration, this paper devotes to development of an efficient control approach for VPSs. The remainder of this paper is organized as follows. Section 2 briefly describes the basic principle of supervisory control theory and the main features of UPPAAL. In Section 3, a two-layer supervisory control architecture based on autonomy and coordination is proposed. On the

L. Li · Z. Jiang (✉)
Department of Industrial Engineering & Management,
School of Mechanical Engineering,
Shanghai Jiao Tong University,
1954 Hua Shan Road,
Shanghai 200030, China
e-mail: zbjiang@mail.sjtu.edu.cn
Tel.: +86-21-62932128

basis of this hybrid architecture, Section 4 establishes the DES model for a VPSs case. To obtain the desired characteristics of performance and activity, autonomous and coordination supervisory controllers are designed for the VPSs case based on heuristic scheduling rules in Section 5. In Section 6, system analysis, calculation and simulation of time-optimal scheduling based on the proposed approach are implemented by UPPAAL. Finally, conclusions and discussions for future work are made in Section 7.

2 Preliminaries

2.1 Basic principle of supervisory control theory

In supervisory control theory, plant G and supervisory controller S construct a supervisory control system S/G in the form of closed loop, as shown in Fig. 1. Firstly, S provides G with the list of authorized events $\Gamma(i)$ (commands) according to the specifications residing in S, where i=0, 1, 2... represents the list index. After receiving these commands, G generates an event $\sigma(i+1) \in (\Sigma_G(q) \cap \Gamma(i))$ and returns it to S as a response, where $\Sigma_G(q)$ is the list of eligible events starting from the current state q in G; and $\sigma(i+1) \in (\Sigma_G(q) \cap \Gamma(i))$ means that event $\sigma(i+1)$ is eligible in the closed-loop supervisory control system if, and only if it is both eligible in G and authorized by S. Then, the response encourages S to enter a new state and commands $\Gamma(i+1)$ starting from the new state are provided for G. Such a process continues until both S and G reach their final states.

2.2 Main features of UPPAAL

As an integrated tool for modeling, simulation, and verification of real-time system, UPPAAL is appropriate for systems that can be modeled as a collection of non-deterministic processes with finite control structure and real-valued clocks, communicating through channels, and (or) shared variables [12]. Its graphical user interface (GUI) is implemented in Java, which consists of three main parts: a system editor, a simulator, and a verifier.

(1) The system editor is used to create and edit the system with timed automata graphically and textually. It is composed of a navigation tree, drawing window, and system description language. The navigation tree is used for accessing the various components of a system description. The drawing window is used for drawing automata through drawing tools. The system description language is a non-deterministic guarded command language with real-valued clock variables and simple data types, which serves as a modeling or design language to describe system behavior as networks of timed automata extended with clock and data variables.
(2) The simulator is a validation tool that enables examination of possible dynamic executions of a system during early design (or modeling) stages and thus provides an inexpensive mean of fault detection prior to verification by the verifier. In addition, the simulator is also used to visualize executions (i.e., traces) generated by the verifier.
(3) The verifier serves as a model checker, which is used to check reachability, safety, and liveness properties by on-the-fly exploration of the state-space of a system in terms of symbolic states represented by constraints. Satisfied properties will be marked green and violated ones red. When trace generation is enabled and a trace is found, the verifier can input the trace into the simulator.

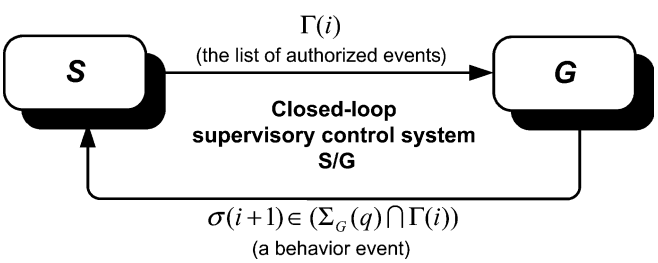


Fig. 1 The basic principle of supervisory control theory

3 Two-layer supervisory control architecture

In the paradigm for VPSs [1], it consists of several simultaneous *VPS*s, and the production resources in each *VPSf* (f=1, 2, ..., m) are logically interrelated on virtual space (control computer) according to the production flow of a specific customized product f. And all the production resources (*RESg*, g=1, 2, ..., n) in VPSs physically belong to several different physical manufacturing systems with different functional structures, e.g., job shops, flexible manufacturing systems (FMSs).

According to its logical and physical characteristics, a two-layer supervisory control architecture based on autonomy and coordination is proposed for VPSs (as shown in Fig. 2), which possesses local quick response of distributed control as well as global optimization of hierarchical control.

The proposed hybrid control structure primarily consists of VPS-layer, resource-layer, and an I/O interface between them. The autonomous supervisory control for each *VPSf* is hierarchically implemented through two kinds of autonomous closed loop and a communication channel. In VPS-layer, each *VPSf* is regarded as a plant G_{VPSf}, which is independently supervised and controlled by its autonomous supervisory controller AS_{VPSf} through the autonomous close loop AS_{VPSf}/G_{VPSf}. The specifications residing in AS_{VPSf} specify the logical relationship (i.e., input and output) between any two resources used in *VPSf*. And in resource-layer, each *RESg* used in *VPSf* is similarly considered as a plant G_{RESg}, which is independently supervised and controlled by its autonomous supervisory controller AS_{RESg} through the autonomous closed loop AS_{RESg}/G_{RESg}. The specifications residing in AS_{RESg} specify the detail operation steps (such as setting up, processing, loading/unloading, and etc.) and time for *RESg*

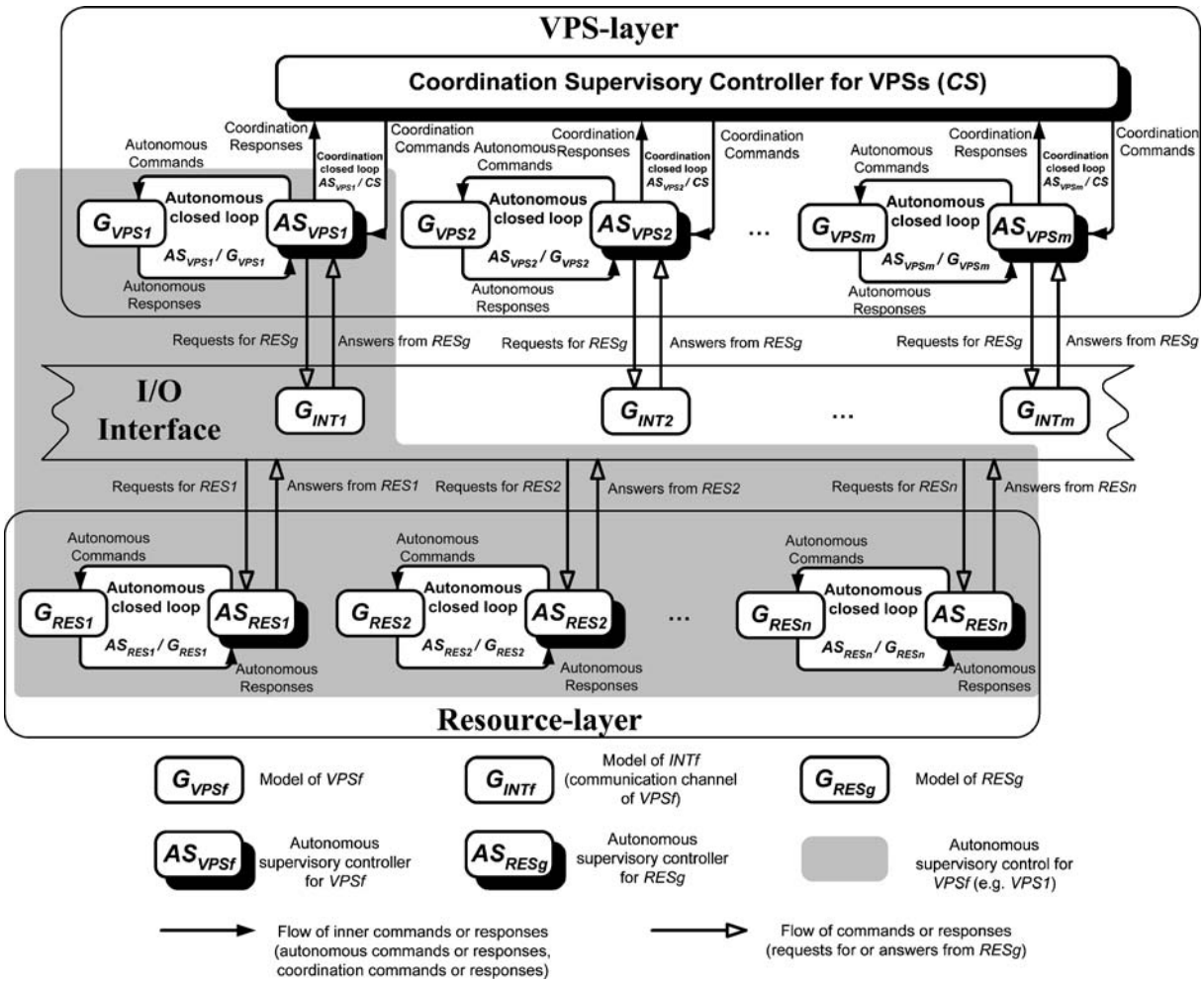


Fig. 2 The two-layer supervisory control architecture based on autonomy and coordination for VPSs

to produce Product *f*. Additionally, the specifications residing in AS_{VPSf} and AS_{RESg} also specify communication mechanisms between the two layers, such that *VPSf* can send requests for or receive answers from *RESg* through its communication channel *INTf* in the I/O interface (here, *INTf* is regarded as a plant G_{INTf}, which is supervised and controlled by both AS_{VPSf} and AS_{RESg}). The coordination supervisory control for multiple *VPS*s running simultaneously is implemented through several coordination closed loops. In VPS-layer, the autonomous control activities of AS_{VPSf} is supervised and controlled by *CS* through the coordination closed loop CS/AS_{VPSf}. The specifications residing in *CS* specify the coordination mechanisms of shared resources.

4 Modeling of VPSs

4.1 A VPSs case

To illustrate how to implement this hybrid supervisory control approach, a VPSs case is considered. The layout of FMS is shown in Fig. 3a, which *(resource ID, c)* denotes a resource and its capacity, and in Fig. 3b, the product production structures (PPSs) of two specific customized products (product *1* and *2*) describe the production tasks by determining the processes, the sequences of the processes, and the estimated time *(T)* /the required resources *(resource ID)* for each process.

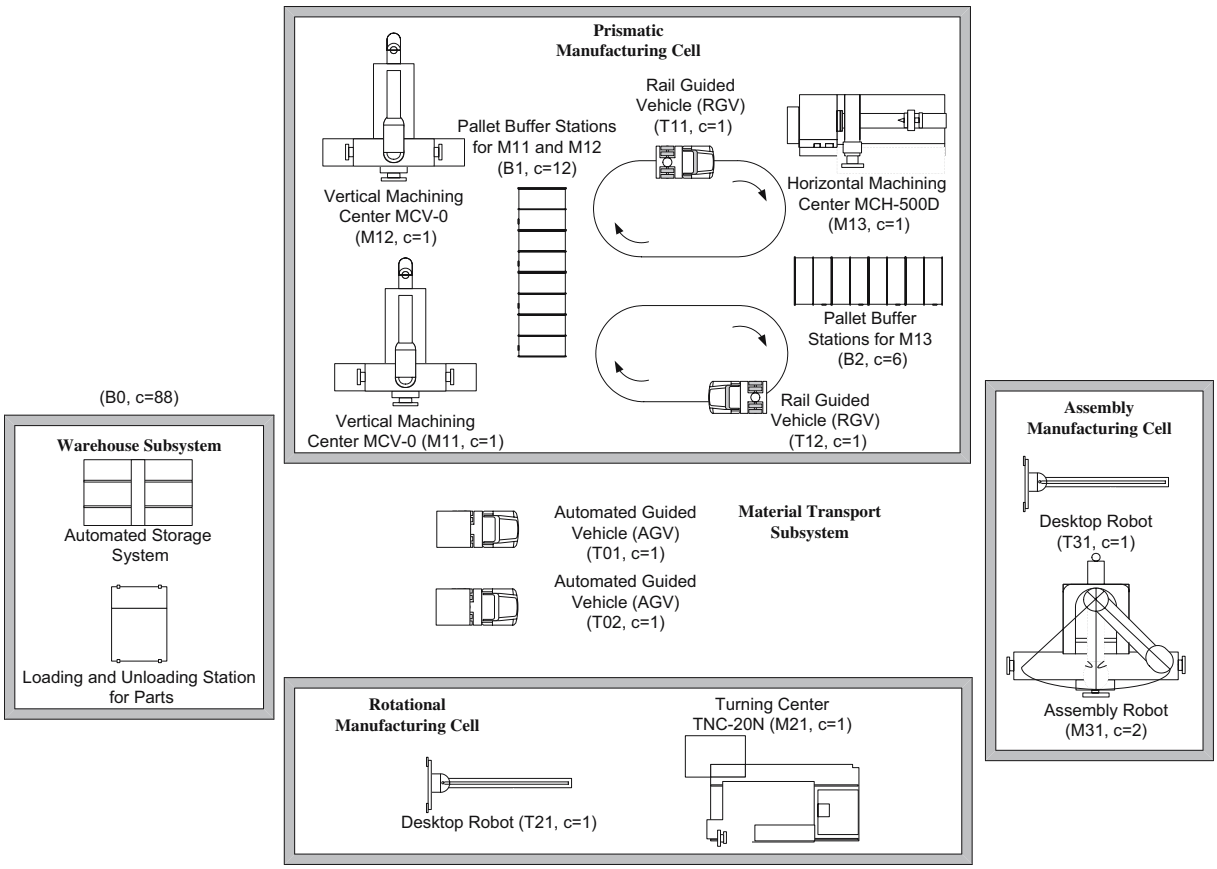
4.2 Modeling of VPSs, resources and communication channels

According to the requirements of modeling, general automata are employed to logically model *VPS*s and communication channels, and timed automata [5] are used to model resources. Uniformly, they can be formulated by following seven-tuple:

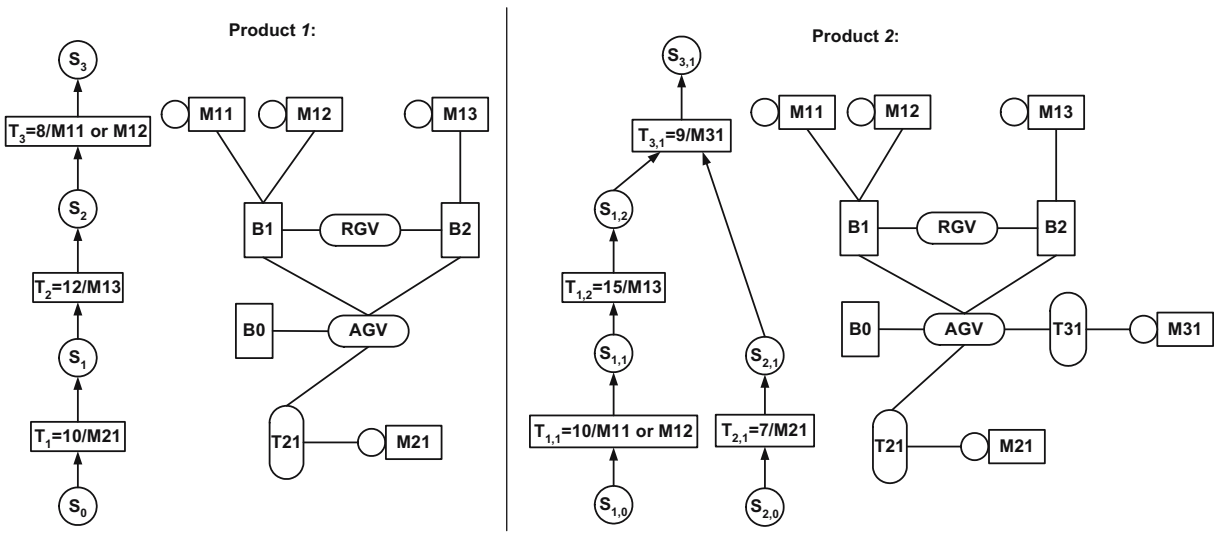
$$G_\Delta = (Q_\Delta, \Sigma_\Delta, \delta_\Delta, q_{\Delta,0}, Q_{\Delta,M}, \Gamma_\Delta, \delta_{\Delta,C}),\ \Delta \in \{VPSf, RESg, INTf\} \quad (1)$$

where,

(1) Q_Δ is the finite set of states, and state $q_\Delta \in Q_\Delta$. Especially for resources, $Q_{RESg} = A_{RESg} \times \prod$



a The layout of a FMS



b The PPSs and VPSs of two case products

Fig. 3 A VPSs case: (**a**) The layout of a FMS; (**b**) The PPSs and VPSs of two case products

$\{T_\sigma | \sigma \in \Sigma_{RESg,act}\}$, where A_{RESg} is the finite set of activity states; $\Sigma_{RESg,act}$ is the finite set of activity events; $T_\sigma = \begin{cases} [0, u_\sigma], 0 \le l_\sigma \le u_\sigma < \infty; \\ [0, l_\sigma], 0 \le l_\sigma < u_\sigma = \infty. \end{cases}$ is the timing interval for event $\sigma \in \Sigma_{RESg,act}$; l_σ is the lower time bound for $\sigma \in \Sigma_{RESg,act}$; u_σ is the upper time bound for $\sigma \in \Sigma_{RESg,act}$.

(2) Σ_Δ is the finite set of events, and event $\omega \in \Sigma_\Delta$. Especially for resources, $\Sigma_{RESg} = \Sigma_{RESg,act} \cup \{tick\}$, where tick is a basic time unit in VPSs, e.g., minutes.

(3) $\delta_\Delta : \Sigma_\Delta \times Q_\Delta \to Q_\Delta$ is the transition function defined by $\delta_\Delta(\omega, q_\Delta) = q'_\Delta$.
(4) $q_{\Delta,0} \in Q_\Delta$ is the initial state.
(5) $Q_{\Delta,M} \subseteq Q_\Delta$ is the set of marker states, and the marker state $q_{\Delta,M} \in Q_{\Delta,M}$. The five basic elements above represent the general characteristics of G_Δ, and the two control elements below describe the controlled mechanisms inside G_Δ.
(6) $\Gamma_\Delta = \{0, 1\}$ is the set of control modes, and $\gamma(\omega) \in \Gamma_\Delta$ is the control mode of event $\omega \in \Sigma_\Delta$. $\gamma(\omega)$=0 if ω is not the authorized event provided by supervisory controller; otherwise, $\gamma(\omega)$=1.
(7) $\delta_{\Delta,C} : \Gamma_\Delta \times \Sigma_\Delta \times Q_\Delta \to Q_\Delta$ is the controlled transition function defined by:

$$\delta_{\Delta,C}(\gamma(\omega) \times \omega \times q_\Delta) = \begin{cases} \delta_\Delta(\omega, q_\Delta) = q'_\Delta, & if\ \gamma(\omega) = 1 \\ undefined, & otherwise \end{cases}. \quad (2)$$

For the purpose of explicit illustration of the VPSs case, the state transition graphs of G_{VPS2}, G_{INT2}, and G_{RESg} (divided into three functional classes: machine, transporter and buffer class) are employed (as shown in Fig. 4), which are the partial automata models of VPSs case. And events in Fig. 4a describe the output and input relationship between any two resources used in *VPS2*.

5 Design of autonomous and coordination supervisory controllers

5.1 Determination of specifications

To obtain the desired characteristics of performance (production flow and time) and activity (overflow-, conflict-, and deadlock-free), autonomous and coordination specifications for VPSs are semantically represented as heuristic scheduling rules.

5.1.1 Autonomous heuristic scheduling rules

Since the autonomous supervisory control for each *VPSf* is hierarchically implemented as mentioned in Section 3, following autonomous heuristic scheduling rules are basically determined.

Rule 1 (for AS_{VPSf}) Each product f must be produced according to the specified production flow, within its production cycle, and by just one production resource at the same time.

Rule 2 (for AS_{RESg}) Each production resource *RESg* must serve each product f in its own production cycle and within the range of its capacity, and each processing step must be accomplished within time interval $[l_\sigma, u_\sigma]$, where l_σ is the earliest finish time and u_σ is the latest finish time.

Rule 3 (communication mechanism for both AS_{VPSf} and AS_{RESg}) Each *VPSf* communicates with its required production resource *RESg* by sending requests to and receiving answers from its communication channel in the I/O interface.

5.1.2 Coordination heuristic scheduling rules

In automatic control of manufacturing systems, overflow, conflict and deadlock are the three basic problems caused by resource sharing. To cope with them, coordination heuristic scheduling rules for *VPS*s can be basically determined.

Rule 1 (overflow-free) If a production resource has already been occupied by products as many as its capacity, then any latter products cannot enter it and have to keep waiting until it is released.

Rule 2 (conflict-free) Each shared production resource adopts first come first serve (FCFS) strategy based on arriving time of tasks and random factors to solve conflicts.

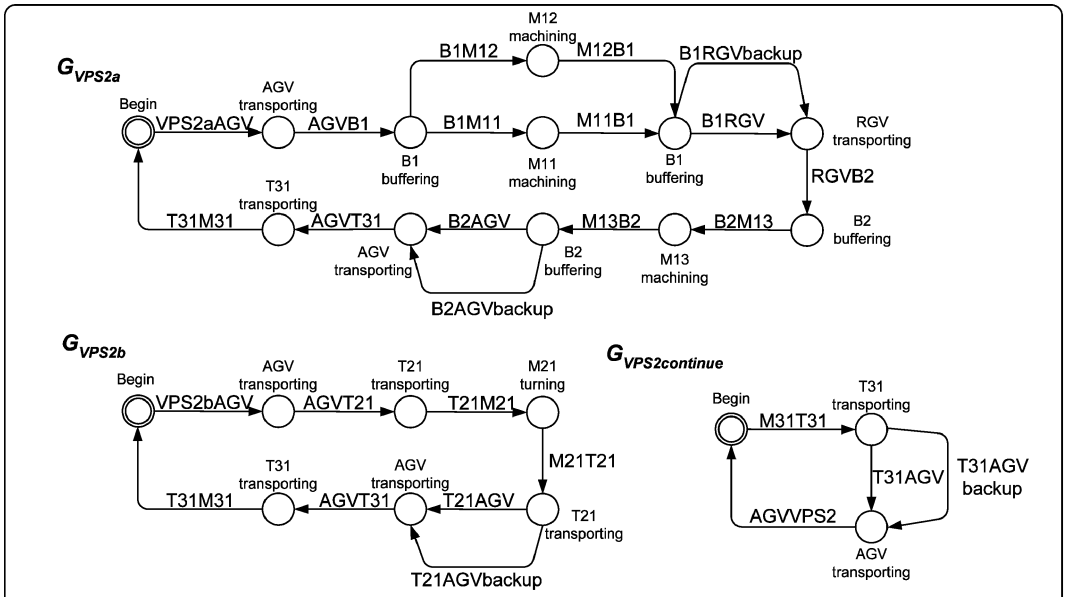
Rule 3 (deadlock-free) (1) Preventive strategy: if deadlock may occur among adjacent production resources, then $\sum_{deadlock} RESg\_Flag > x - d$, where x is the amount of these resources, d is the amount of deadlocks existing among them, and $\sum_{deadlock} RESg\_Flag$ is the amount of their total spare spaces; (2) Backup strategy: if deadlock occurs between AGV/RGV and its adjacent resource, then backup AGV/RGV will be used as a buffer. Firstly, the product in the machine will move to the backup AGV/RGV and the machine is released. Then the product in the AGV/RGV will move to the machine. Finally, backup AGV/RGV will change to AGV/RGV and original AGV/RGV will change to backup AGV/RGV.

5.2 Formalization of specifications

5.2.1 Autonomous supervisory controllers based on autonomous rules

With the formalization of the above autonomous specifications through their combination with G_{VPSf}, G_{RESg}, and G_{INTf}, the autonomous supervisory controllers are formally designed as $AS_\Theta = (X_{AS,\Theta}, \Sigma_{AS,\Theta}, \xi_{AS,\Theta}, x_{AS,\Theta,0}, X_{AS,\Theta,M})$, $\Theta \in \{VPSf, RESg\}$, where five elements is similar to those in Eq. 1.

Figure 5 shows the state transition graphs of AS_{VPS2} and AS_{RESg}, where events *goVPSf*, *CAGVT21*, and etc. belong to ΣCS(the finite events set of coordination supervisory controller and will be mentioned later), through which the coordination controller can logically coordinate the auton-



$\Sigma_{VPS2} = \Sigma_{VPS2a} \cup \Sigma_{VPS2b} \cup \Sigma_{VPS2continue}$

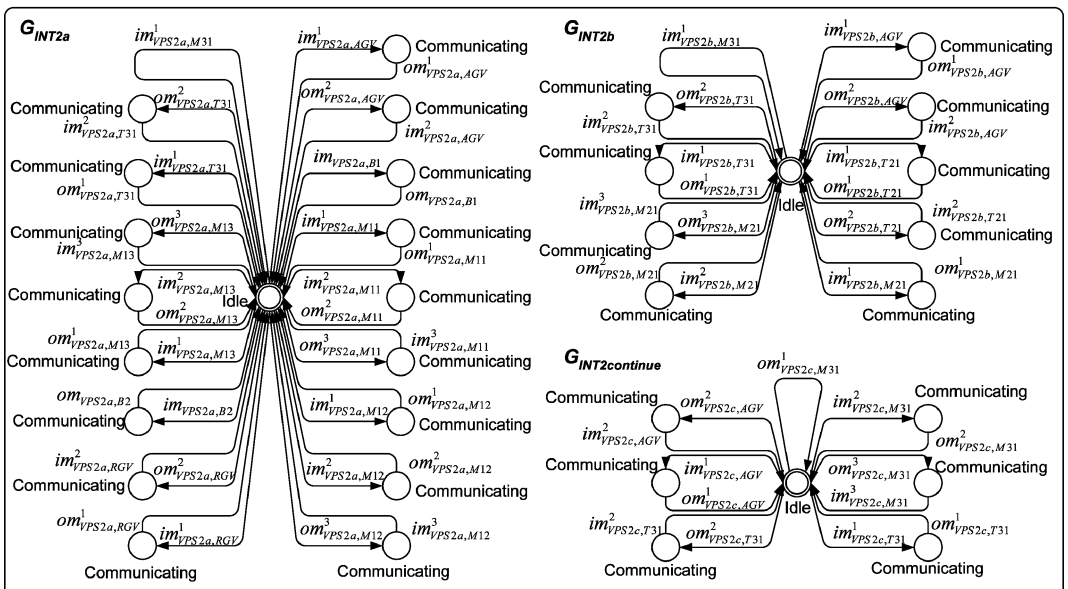
= {VPS2aAGV, AGVB1, B1M11, M11B1, B1M12, M12B1, B1RGV, B1RGVbackup, RGVB2, B2M13, M13B2, B2AGV, B2AGVbackup, VPS2bAGV, AGVT21, T21M21, M21T21, T21AGV, T21AGVbackup, AGVT31, T31M31, M31T31, T31AGV, T31AGVbackup, AGVVPS2}

$Q_{VPS2} = Q_{VPS2a} \cup Q_{VPS2b} \cup Q_{VPS2continue}$

= {Begin, AGVtransporting, B1buffering, M11machining, M12machining, RGVtransporing, B2buffering, M13machining, T21transporting, M21turning, T31transporting}

$q_{VPS2,0} = q_{VPS2,M} =$ Begin

a The state transition graph of G_{VPS2}



$\Sigma_{INT2} = \Sigma_{INT2a} \cup \Sigma_{INT2b} \cup \Sigma_{INT2continue}$

$= \{im^1_{VPS2a,AGV}, om^1_{VPS2a,AGV}, im^2_{VPS2a,AGV}, om^2_{VPS2a,AGV}, im_{VPS2a,B1}, om_{VPS2a,B1}, im^1_{VPS2a,M11}, om^1_{VPS2a,M11}, im^2_{VPS2a,M11}, om^2_{VPS2a,M11},$
$im^3_{VPS2a,M11}, om^3_{VPS2a,M11}, im^1_{VPS2a,M12}, om^1_{VPS2a,M12}, im^2_{VPS2a,M12}, om^2_{VPS2a,M12}, im^3_{VPS2a,M12}, om^3_{VPS2a,M12}, im^1_{VPS2a,RGV}, om^1_{VPS2a,RGV},$
$im^2_{VPS2a,RGV}, om^2_{VPS2a,RGV}, im_{VPS2a,B2}, om_{VPS2a,B2}, im^1_{VPS2a,M13}, om^1_{VPS2a,M13}, im^2_{VPS2a,M13}, om^2_{VPS2a,M13}, im^3_{VPS2a,M13}, om^3_{VPS2a,M13},$
$im^1_{VPS2a,T31}, om^1_{VPS2a,T31}, im^2_{VPS2a,T31}, om^2_{VPS2a,T31}, im^1_{VPS2a,M31}, im^1_{VPS2b,AGV}, om^1_{VPS2b,AGV}, im^2_{VPS2b,AGV}, om^2_{VP2b,AGV},$
$im^1_{VPS2b,T21}, om^1_{VPS2b,T21}, im^2_{VPS2b,T21}, om^2_{VPS2b,T21}, im^1_{VPS2b,M21}, om^1_{VPS2b,M21}, im^2_{VPS2b,M21}, om^2_{VPS2b,M21}, im^3_{VPS2b,M21}, om^3_{VPS2b,M21},$
$im^1_{VPS2b,T31}, om^1_{VPS2b,T31}, im^2_{VPS2b,T31}, om^2_{VPS2b,T31}, im^1_{VPS2b,M31}, om^1_{VPS2c,M31}, im^2_{VPS2c,M31}, om^2_{VPS2c,m31}, im^3_{VPS2c,M31}, om^3_{VPS2c,M31},$
$im^1_{VPS2c,T31}, om^1_{VPS2c,T31}, im^2_{VPS2c,T31}, om^2_{VPS2c,T31}, im^1_{VPS2c,AGV}, om^1_{VPS2c,AGV}, im^2_{VPS2c,AGV}, om^2_{VPS2c,AGV}\}$

$Q_{INT2} = Q_{INT2a} \cup Q_{INT2b} \cup Q_{INT2continue} =$ {Idle, Communicating} $q_{INT2,0} = q_{INT2,M} =$ Idle

Fig. 4 (continued)

WHEN *RESg*={M11, M12, M13, M21, M31}, $im^k_{VPSf,RESg}$: { A request for setting up *RESg* for *VPSf*, if k=1; A request for processing *VPSf* on *RESg*, if k=2; A request for repairing *RESg* for *VPSf*, if k=3

AND $om^k_{VPSf,RESg}$: { An answer of completing setting up *RESg* for *VPSf*, if k=1; An answer of completing processing *VPSf* on *RESg*, if k=2; A message that *RESg* breaks down when processing *VPSf*, if k=3

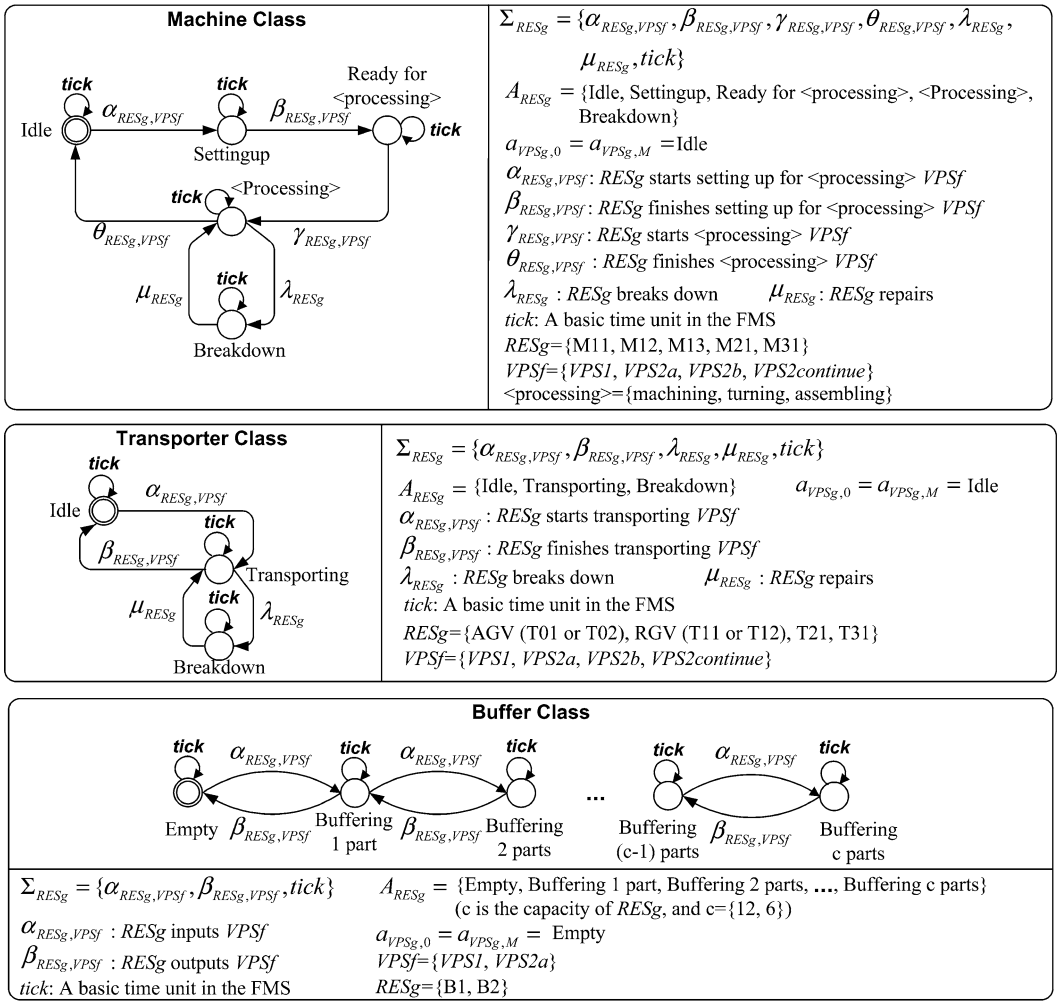
WHEN *RESg*={AGV (T01 or T02), RGV (T11 or T12), T21, T31}, $im^k_{VPSf,RESg}$: { A request for transporting *VPSf* on *RESg*, if k=1; A request for repairing *RESg* for *VPSf*, if k=2

AND $om^k_{VPSf,RESg}$: { An answer of completing transporting *VPSf* on *RESg*, if k=1; A message that *RESg* breaks down when transporting *VPSf*, if k=2

WHEN *RESg*={B1, B2}, $im_{VPSf,RESg}$: A request for inputting *VPSf* into *RESg*

AND $om_{VPSf,RESg}$: An message of outputting *VPSf* from *RESg*

b The state transition graph of G_{INT2}



Machine Class

$\Sigma_{RESg} = \{\alpha_{RESg,VPSf}, \beta_{RESg,VPSf}, \gamma_{RESg,VPSf}, \theta_{RESg,VPSf}, \lambda_{RESg}, \mu_{RESg}, tick\}$

A_{RESg} = {Idle, Settingup, Ready for <processing>, <Processing>, Breakdown}

$a_{VPSg,0} = a_{VPSg,M}$ =Idle

$\alpha_{RESg,VPSf}$: *RESg* starts setting up for <processing> *VPSf*

$\beta_{RESg,VPSf}$: *RESg* finishes setting up for <processing> *VPSf*

$\gamma_{RESg,VPSf}$: *RESg* starts <processing> *VPSf*

$\theta_{RESg,VPSf}$: *RESg* finishes <processing> *VPSf*

λ_{RESg} : *RESg* breaks down μ_{RESg} : *RESg* repairs

tick: A basic time unit in the FMS

RESg={M11, M12, M13, M21, M31}

VPSf={*VPS1*, *VPS2a*, *VPS2b*, *VPS2continue*}

<processing>={machining, turning, assembling}

Transporter Class

$\Sigma_{RESg} = \{\alpha_{RESg,VPSf}, \beta_{RESg,VPSf}, \lambda_{RESg}, \mu_{RESg}, tick\}$

A_{RESg} = {Idle, Transporting, Breakdown} $a_{VPSg,0} = a_{VPSg,M}$ = Idle

$\alpha_{RESg,VPSf}$: *RESg* starts transporting *VPSf*

$\beta_{RESg,VPSf}$: *RESg* finishes transporting *VPSf*

λ_{RESg} : *RESg* breaks down μ_{RESg} : *RESg* repairs

tick: A basic time unit in the FMS

RESg={AGV (T01 or T02), RGV (T11 or T12), T21, T31}

VPSf={*VPS1*, *VPS2a*, *VPS2b*, *VPS2continue*}

Buffer Class

$\Sigma_{RESg} = \{\alpha_{RESg,VPSf}, \beta_{RESg,VPSf}, tick\}$

$\alpha_{RESg,VPSf}$: *RESg* inputs *VPSf*

$\beta_{RESg,VPSf}$: *RESg* outputs *VPSf*

tick: A basic time unit in the FMS

A_{RESg} = {Empty, Buffering 1 part, Buffering 2 parts, ..., Buffering c parts} (c is the capacity of *RESg*, and c={12, 6})

$a_{VPSg,0} = a_{VPSg,M}$ = Empty

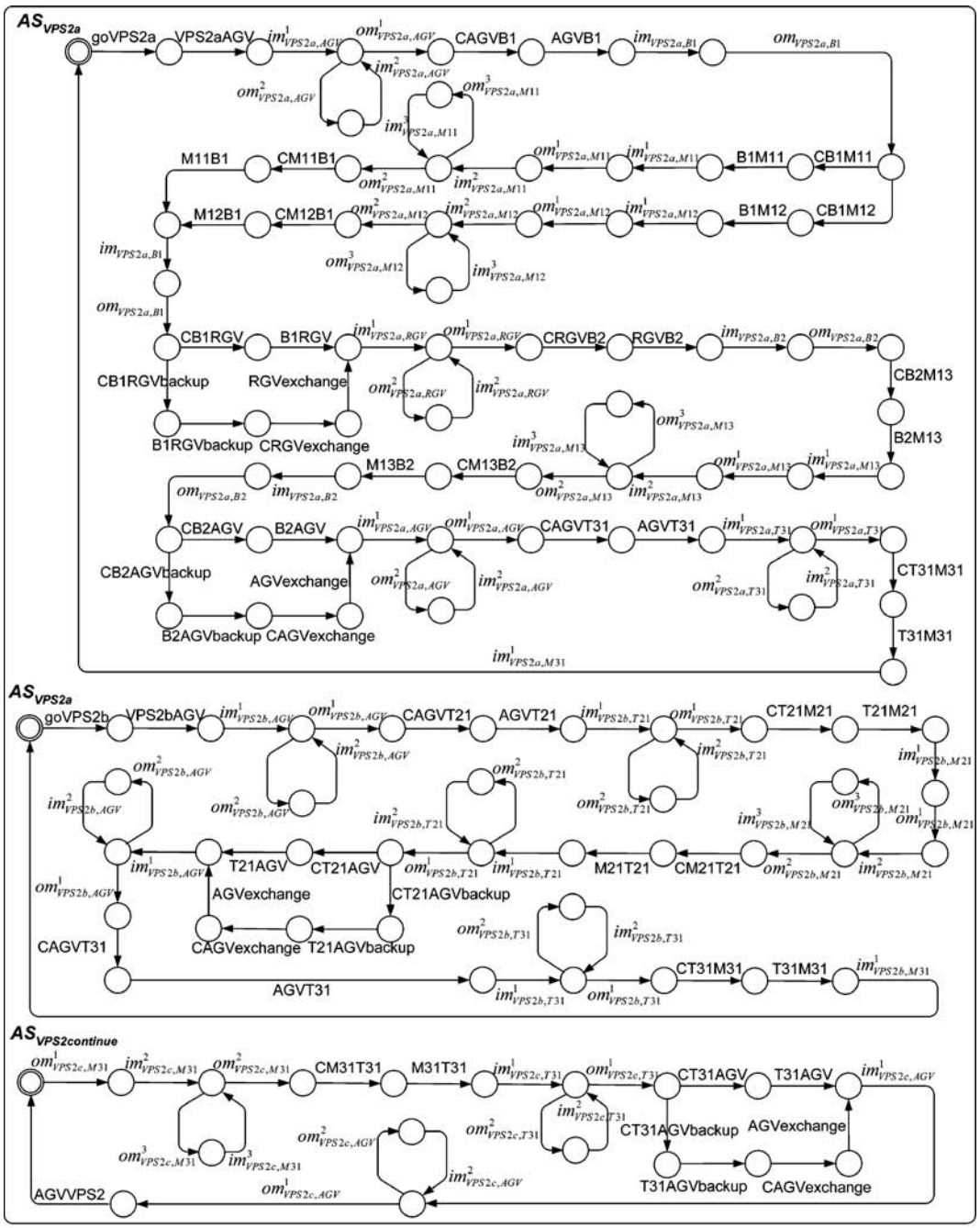
VPSf={*VPS1*, *VPS2a*}

RESg={B1, B2}

c The state transition graphs of G_{RESg}

◀ **Fig. 4** The partial automata models of VPSs case: (**a**) The state transition graphs of G_{VPS2}; (**b**) The state transition graphs of G_{INT2}; and (**c**) The state transition graphs of G_{RES}

omous control activities of AS_{VPS2} with that of AS_{VPS1} before they are actually implemented in *VPS2*; events *VPSfAGV*, *AGV21*, and etc. belong to Σ_{VPSf}, through which AS_{VPS2} can supervise and control *VPS2* independently; events $\alpha_{RESg,VPSf}$, $\beta_{RESg,VPSf}$, and etc. belong to Σ_{RESg}, through which AS_{RESg} can supervise and control *RESg* independently; and events $im^k_{VPSf,RESg}$ and $om^k_{VPSf,RESg}$ belong to Σ_{INTf}, through which AS_{VPS2} can make communication with AS_{RESg} and realize the hierarchical control.



a The state transition graph of AS_{VPS2}

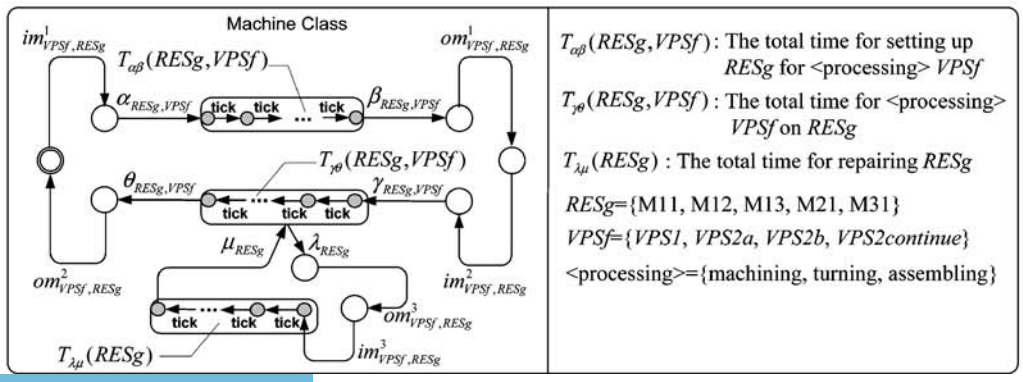
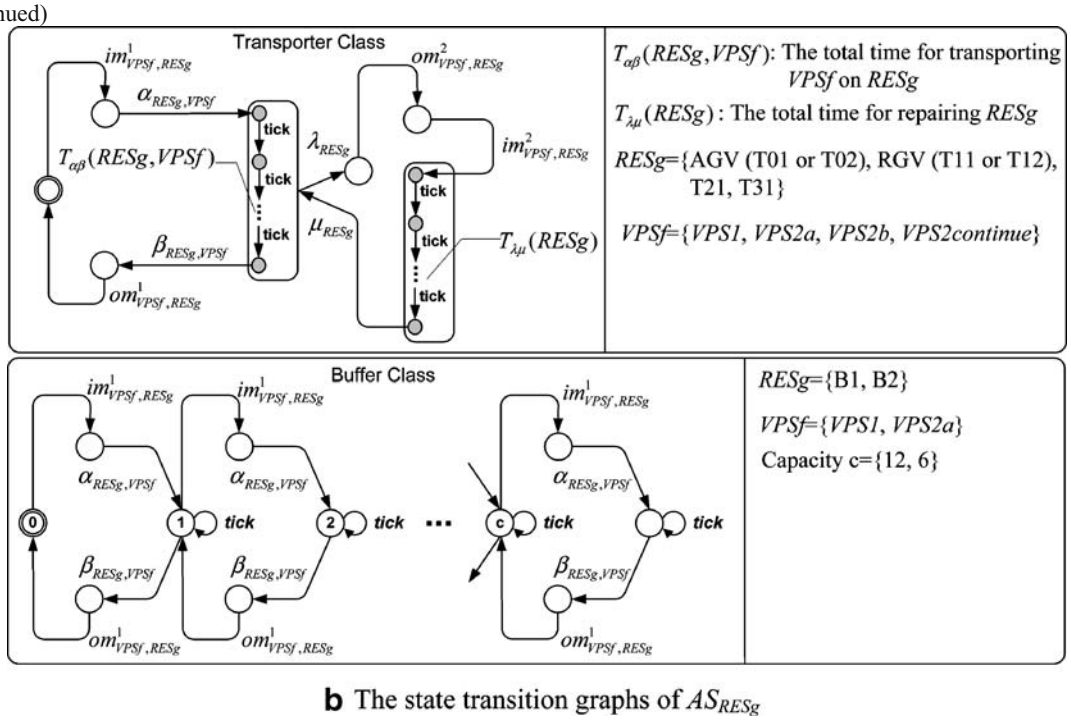


Fig. 5 (continued)



b The state transition graphs of AS_{RESg}

◀ **Fig. 5** The partial automata models of autonomous supervisory controllers: (**a**) The state transition graph of AS_{VPS2}; (**b**) The state transition graphs of AS_{RESg}

5.2.2 Coordination supervisory controllers based on coordination rules

Motivated by that the coordination supervisory controller must be able to supervise and control multiple *VPS*s simultaneously, a modified finite capacity machine is proposed to design the coordination supervisory controller.

$$CS = \left(X_{CS}, X'_{CS}, X_{CS,t}, \Sigma_{CS}, \mathrm{B}(e), \mathrm{P}(e), \xi_{CS}, x_{CS,0}, X_{CS,M}\right) \tag{3}$$

where,

(1) X_{CS} is the finite set of locations. Location $x_{CS,g}\in X_{CS}$ indicates RESg and has two state parameters: pg (the total amount of digitized tokens located in $x_{CS,g}$) and cg (the capacity of $x_{CS,g}$), in which $pg\le cg$ and a digitized token represents a VPS;

(2) $X'_{CS}=\{0,1\}$ is the set of location-availability states, which describes the availability of resources. If $x_{CS,g}$ is unavailable for a transition from other locations ($pg=cg$ or RESg breaks down), then $x'_{CS}(x_{CS,g})=0$ otherwise, $x'_{CS}(x_{CS,g})=1$;

(3) $X_{CS,t}=\times_{f=1}^{m}X^{f}_{CS,t}$ is the set of token-position of multiple digitized tokens, which describes the position of VPSs, where $X^{f}_{CS,t}=\{0,1\}$ is the set of token-position of the fth digitized token. If the fth digitized token is not in location $x_{CS,g}$, then $x^{f}_{CS,t}(x_{CS,g})=0$; otherwise, $x^{f}_{CS,t}(x_{CS,g})=1$;

(4) Σ_{CS} is the finite set of events;

(5) $\mathrm{B}(e)=\times_{f=1}^{m}\mathrm{B}(e_f)$ is the set of transition guards of multiple digitized tokens, where $\mathrm{B}(e_f)$ is the set of transition guards of the fth digitized token and guard $u_f\in\mathrm{B}(e_f)$;

(6) $\mathrm{P}(e)=\times_{f=1}^{m}\mathrm{P}(e_f)$ is the set of transition updates of multiple digitized tokens, where $\mathrm{P}(e_f)$ is the set of transition updates of the fth digitized token and update $v_f\in\mathrm{P}(e_f)$;

(7) $\xi_{CS}:\mathrm{B}(e)\times\Sigma_{VPSf}\times X_{CS}\times\mathrm{P}(e)\to X_{CS}$ is the transition function of the fth digitized token, which is defined by

$$\xi_{CS}(u_f,\sigma_f,x_{CS,g},v_f)=\begin{cases}\xi_{CS}(\sigma_f,x_{CS,g},v_f)\ x_{CS,g'} & \text{if } u_f \text{ is satisfied;}\\ \text{undefined}, & \text{otherwise.}\end{cases} \tag{4}$$

(8) $X_{CS,0}\in X_{CS}$ is the initial location;

(9) $X_{CS,M}\subseteq X_{CS}$ is the set of marker locations, and marker location $x_{CS,M}\in X_{CS,M}$.

Comparing with the finite capacity machine (FCM) defined in [6], the modified FCM has a decision-making function by adding two sets of control: (1) the set of transition guards, and (2) the set of transition updates. Formally, a transition guard determines the transitions conditions through several constraints in the form $e\sim z$ for $z\in Z^{+}\cup\{0\}$ and $\sim\in\{<,<=,==,>,>\}$, where e is a variable or an expression, and Z^{+} is the set of positive integers. And a transition update determines the changes after firing

transitions through several value assignments in the form $e{:=}z$.

The state transition graph of *CS* for the VPSs case is shown in Fig. 6, where Σ_{CS} is corresponding to that in AS_{VPS1} and AS_{VPS2} and the usage of integer variables as heuristic functions is the key of coordination (as listed in the Table 1).

On one hand, they define the occurrence conditions of autonomous control activities, which are the decision-making standards for coordination. On the other hand, they continuously update according to the actual implementation of autonomous control activities, which are the evidences for the next coordination. Additionally, a clock *total* is used to record a cycle time of VPSs, i.e. make-span.

6 Implementation of system analysis and optimal scheduling

6.1 Deadlock-free analysis

The verifier in UPPAAL can check whether the system satisfies the desired properties by queries. Since deadlock-free means the state of deadlock will never happen, i.e., the state of not deadlock will always be true, query **A[] not deadlock** is used to make deadlock-free analysis for systems, in which path formula A[] denotes that the its following state formula is true in all reachable states. Through verification, it is proved that the VPSs case under the control of proposed approach is deadlock-free.

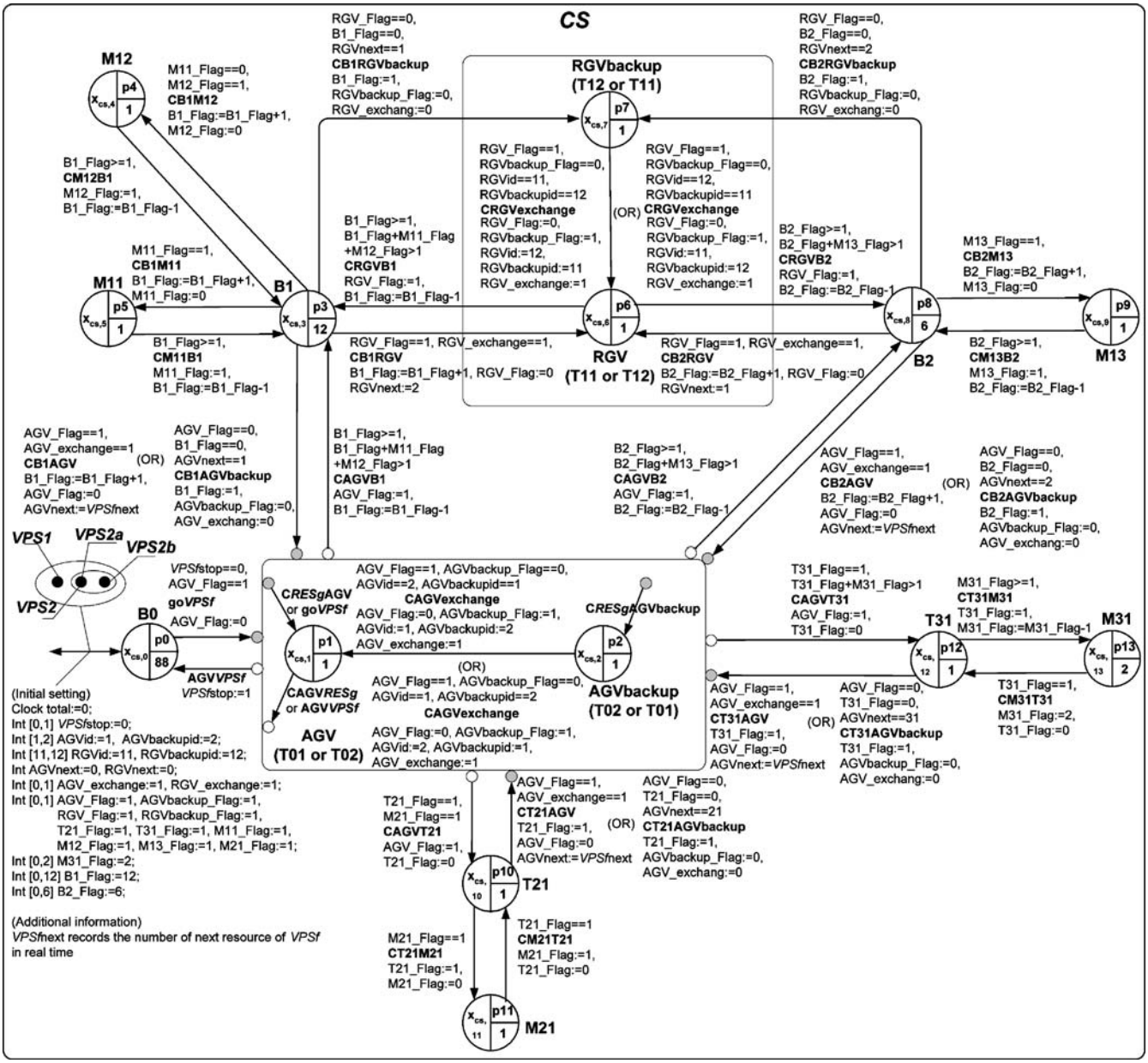


Fig. 6 The automata model of coordination supervisory controller

Table 1 Integer variables in coordination supervisory controller of VPSs

Integer variables	Meanings	values
ResourceID_Flag(Equalsto cg-pg)	The spare space of resources	Non-negative integer
AGVnext,RGVnext	The resource that AGV/RGV next to arrive	1: B1; 2: B2; 21: T21; 31: T31
AGV_exchange,RGV_exchange	The exchanging progress between AGV/RGV and AGVbackup/ RGVbackup	0: begin exchanging; 1: finish exchanging
AGVid,AGVbackupid,RGVid, RGVbackupid	The current resource ID of AGV, AGVbackup, RGV and RGVbackup	1: *T01*; 2: *T02*; 11:*T11*; 12: *T12*
VPSfstop	The progress of *VPSf*	0: not complete; 1: complete

6.2 Calculation and simulation of time-optimal scheduling

In UPPAAL, a trace can be calculated according to a diagnostic trace strategy by the verifier and be simulated by the simulator. To calculate the time-optimal scheduling for the VPSs case, query **E<>(VPS1.Begin and VPS2a.Begin and VPS2b.Begin and VPS2continue.Beginand initial-module.Stop)** and fastest strategy is used to find a trace starting at the initial state, ending at the desired state and owning optimal make-span, in which path formula E<> denotes that its following state formula is reachable eventually. As a result, the make-span of time-optimal scheduling is 61 ticks and the simulation is shown in Fig. 7.

Compared with the existing scheduling methodologies [13, 14], the proposed supervisory control approach integrates control and scheduling into one supervisory control system, such that the activities of VPSs can be completely under control (overflow-, conflict-, deadlock-free) when the calculated optimal scheduling operates in practice. Therefore, this paves the way to obtain practical results from optimal scheduling.

7 Conclusions

In this paper, we have proposed a hybrid supervisory control approach based on the mechanism of autonomy and coordination by using supervisory control theory, through

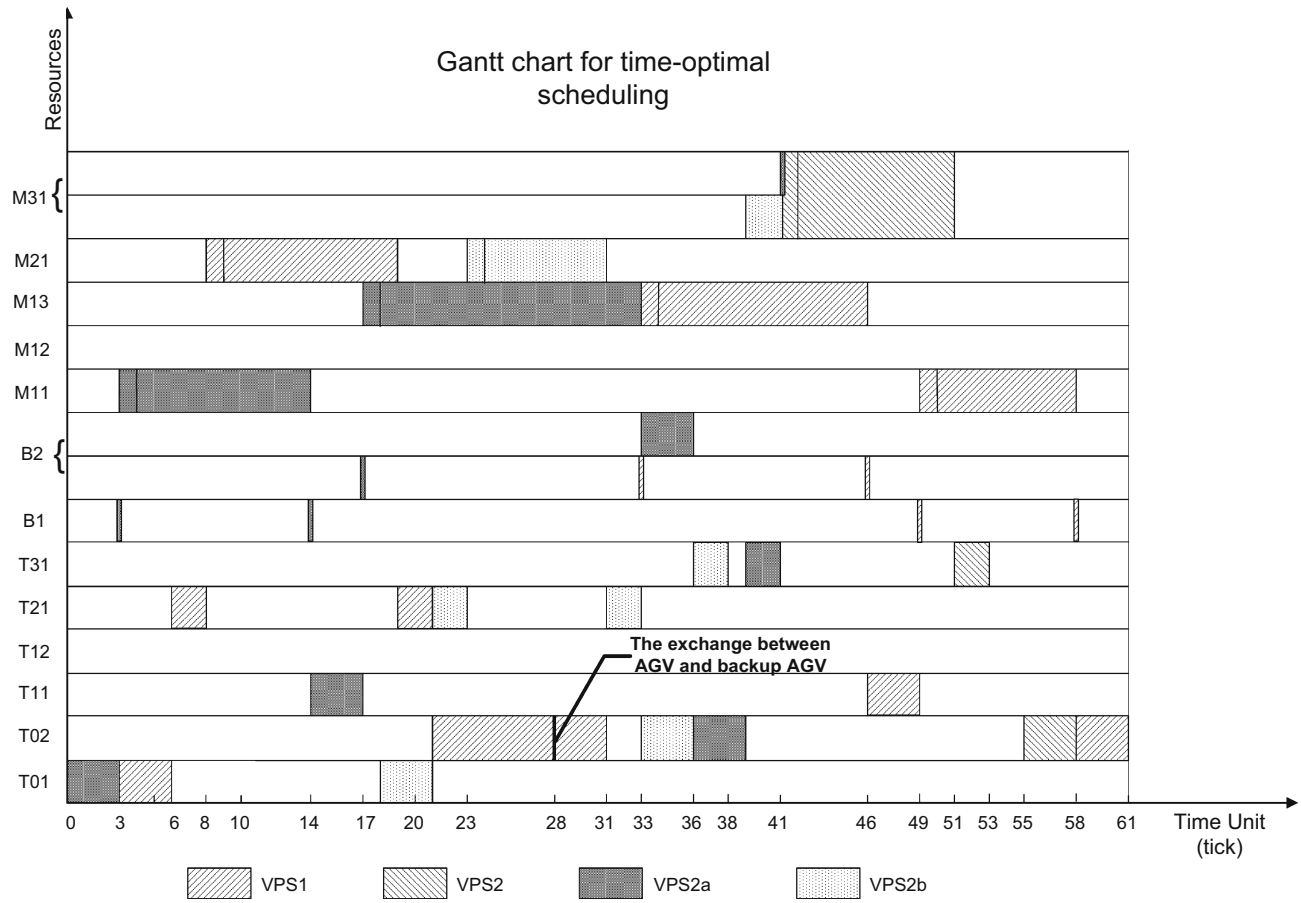


Fig. 7 Simulation of time-optimal scheduling for the VPSs case

which both control and scheduling for VPSs can be achieved.

Firstly, the control structure combines the advantages of both distributed and hierarchical control, in which both autonomous supervisory control for each *VPS* and coordination supervisory control for multiple *VPS*s can be implemented. Secondly, autonomous and coordination supervisory controllers are designed based on the heuristic scheduling rules, and a modified FCM is especially developed for the coordination supervisory controller. Finally, system analysis, calculation and simulation of time-optimal scheduling for a VPSs case based on the proposed approach are successfully implemented by UPPAAL, which proved its implementability and effectiveness.

In order to extend the proposed approach into adaptively dynamic control of VPSs, adaptive scheduling mechanisms and corresponding expert system will be studied in the future, such that an integrated and intelligent supervisory control environment of VPSs can be established systemically.

Acknowledgement The work described in this paper was substantially supported by a Research Grant from the National Natural Science Foundation of China (70271036).

References

1. Jiang ZB, Fung RYK, Tu YL et al (2000) A framework for adaptive control of virtual production systems. Proc 3rd World Congress on Intelligent Control and Automation (WCICA'2000), IEEE Press, pp138–142
2. Fung RYK, Jiang ZB, Zou MJ et al (2002) Adaptive production scheduling of virtual production systems using OPNS-CS with changeable structure. Int J Prod Res 40(8):1759–1785
3. Jiang ZB, Fung RYK (2003) An adaptive agile manufacturing control infrastructure based on TOPNs-CS modeling. Int J Adv Manuf Technol 22:191–215
4. Desrochers AA, Al-Jaar RY (1995) Applications of Petri nets in manufacturing systems. IEEE, New York
5. Brandin BA, Wonham WM (1994) Supervisory control of timed discrete-event systems. IEEE Trans 39(2):329–342
6. Qiu RG, Joshi SB (1999) A structured adaptive supervisory control methodology for modeling the control of a discrete event manufacturing system. IEEE Trans Syst, Man Cybern, Part A: Syst Hum 29(6):573–586
7. Leduc RJ, Brandin BA, Wonham WM et al (2001) Hierarchical interface-based supervisory control: serial case. Proc. 40th IEEE Conf Decis Control 5:4116–4121
8. Leduc RJ, Wonham WM, Lawford M (2002) Hierarchical interface-based supervisory control: parallel case. Proc 39th Allerton Conference on Comm, Contr, and Comp, pp 386–395
9. Chandra V, Oruganti B, Kumar R (2002) UKDES: a graphical software tool for the design, analysis & control discrete event systems. IEEE Trans Control Syst Technol, submitted for publication
10. Fabian M, Hellgren A (2000) Desco - a tool for education and control of discrete event systems. Kluwer, Dordrecht
11. UPPAAL, http://www.uppaal.com/
12. Behrmann G, David A, Larsen KG (2001) UPPAAL-present and future. Proc 40th IEEE Conference on Decision and Control (CDC'2001), Orlando, FL, USA, pp 2881–2886
13. Cavalieri S, Mirabella O (1996) A PN-based scheduler for a flexible semiconductor manufacturing system. Proc 1996 IEEE Conference on Emerging Technologies and Factory Automation (ETFA'96), Kauai, HI USA, pp 724–729
14. Reyes A, Yu H, Lloyd S (2001) An evolutionary hybrid scheduler based in Petri net structures for FMS scheduling. Proc 2001 IEEE International Conference on Systems, Man, and Cybernetics, Tucson, AZ USA, pp 2516–2521