On Scalable Supervisory Control of Multi-Agent Discrete-Event Systems

Yingying Liu, Kai Cai, and Zhiwu Li (August 15, 2017)

Abstract—In this paper we study multi-agent discrete-event systems where the agents can be divided into several groups, and within each group the agents have similar or identical state transition structures. We employ a relabeling map to generate a “template structure” for each group, and synthesize a scalable supervisor whose state size and computational process are independent of the number of agents. This scalability allows the supervisor to remain invariant (no recomputation or reconfiguration needed) if and when there are agents removed due to failure or added for increasing productivity. The constant computational effort for synthesizing the scalable supervisor also makes our method promising for handling large-scale multi-agent systems. Moreover, based on the scalable supervisor we design scalable local controllers, one for each component agent, to establish a purely distributed control architecture. Three examples are provided to illustrate our proposed scalable supervisory synthesis and the resulting scalable supervisors as well as local controllers.

I. INTRODUCTION

Multi-agent systems have found increasing applications in large-scale engineering practice where tasks are difficult to be accomplished by a single entity. Examples include multiple machines in factories, robots in manufacturing cells, and AGVs in logistic systems [1], [2], [3]. Although not always the case, multi-agent systems typically can be divided into several groups, according to different roles, functions, or capabilities. For instance, machines are grouped to process different types of workpieces, robots to manufacture different parts of a product, AGVs to transport items of distinct sizes, shapes and weights. Agents in the same group often have similar or even identical state transition structures, i.e. dynamics. This we shall refer to as a modular characteristic.

In this paper we study multi-agent systems with such a modular characteristic, and consider individual agents modeled by discrete-event systems (DES). Given a control specification, one may in principle apply supervisory control theory [4], [5], [6] to synthesize a monolithic (i.e. centralized) supervisor for the entire multi-agent system. While the supervisor computed by this method is optimal (i.e. maximally permissive) and nonblocking, there are two main problems. First, the state size of the supervisor increases (exponentially) as the number of agents increases [7]; consequently, the supervisor synthesis will become computationally infeasible for large numbers of agents. Second, whenever the number of agents changes (increases when more agents are added into the system to enhance productivity or to improve redundancy for the sake of reliability; or decreases when some agents malfunction and are removed from the system), the supervisor must be recomputed or reconfigured (e.g. [8], [9]) in order to adapt to the change.

The first problem may be resolved by decentralized and/or hierarchical supervisory synthesis methods (e.g. [10], [11], [12], [13]). These methods, however, usually can deal only with fixed numbers of agents, and thus must also be recomputed or reconfigured if and when the agent number changes.

In this paper we solve both problems mentioned above by exploiting the modular characteristic of multi-agent systems, and thereby designing a scalable supervisor whose state number and computational process are independent of the number of agents. First, owing to similar/identical state transition structures of agents in the same group, we employ a relabeling map (precise definition given in Section II.A below) to generate a “template structure” for each group. The template structures thus generated are independent of the agent numbers. Then we design a supervisor based on these template structures, and prove that it is a scalable supervisor for the multi-agent system under certain sufficient conditions. The controlled behavior of the designed scalable supervisor need not be optimal, but is nonblocking. Moreover, we show that the sufficient conditions for the scalable supervisor are efficiently checkable.

While the designed scalable supervisor serves as a centralized controller for the multi-agent system, it may sometimes be natural, and even more desirable, to equip each individual agent with its own local controller (such that it becomes an autonomous, intelligent agent). Hence we move on to design scalable local controllers whose state numbers and computational process are invariant with respect to the number of component agents; for this design, we employ the method of supervisor localization [14], [15], [16]. Directly localizing the scalable supervisor may be computationally expensive, inasmuch as the localization method requires computing the entire plant model. To circumvent this problem, we localize the supervisor based on the template structures and thereby derive scalable local controllers without constructing the underlying plant model. It is proved that the collective controlled behavior of these local controllers is equivalent to that achieved by the scalable supervisor.

The contributions of our work are threefold. First, our designed centralized supervisor has scalability with respect to the number of agents in the system. This scalability is a desired feature of a supervisor for multi-agent systems,
inasmuch as it allows the supervisor to remain invariant regardless of how many agents are added to or removed from the system (which may occur frequently due to productivity/reliability concerns or malfunction/repair). Second, the local controllers we designed for individual agents have the same scalability feature, and are guaranteed to collectivity achieve identical controlled behavior as the centralized supervisor does. With the local controllers ‘built-in’, the agents become autonomous and make their own local decisions; this is particularly useful in applications like multi-robot systems. Finally, the computation of the scalable supervisor and local controllers is based solely on template structures and is thus independent of agent numbers as well. As a result, the computation load remains the same even if the number of agents increases; this is advantageous as compared to centralized/decentralized supervisory synthesis methods.

We note that [17] also studied multi-agent systems with a modular characteristic and used group-theoretic tools to characterize symmetry among agents with similar/identical structures. Exploiting symmetry, “quotient automata” were constructed to reduce the state size of the composed system, based on which supervisors are synthesized. Quotient automata construction was further employed in [18] to develop decentralized synthesis and verification algorithms for multi-agent systems. While the systems considered in [17], [18] are more general than ours in that agents are allowed to share events, the state size of the resulting quotient automata is dependent on the agent numbers and in the worst case exponential in the number of agents. By contrast, we use the relabeling map approach and synthesize scalable supervisors whose state sizes are independent of agent numbers.

We also note that in [19], an automaton-based modeling framework was presented for multi-agent systems in which the agents’ dynamics are instantiated from a finite number of “templates”; a particular product operation enforcing synchronization on broadcasting or receiving events was proposed to compose the agent dynamics. Building on [19], the work in [20] proposed a method that first decomposes the overall control specification into local ones for individual agents, and then incrementally synthesizes a supervisor based on the local specifications. The presented algorithm for incremental synthesis is (again) dependent on, and in general exponential in, the number of agents.

By extending the ideas in [19] and [20], the work in [21] proposed a scalable control design for a type of multi-agent systems, where an “agent” was not just a plant component, but indeed a plant of its own including an imposed specification. The “agents” were instantiated from a template; for the template, under certain conditions, an algorithm was proposed to design a supervisor whose instantiation was shown to work for each “agent”. By contrast, we consider multi-agent systems where each agent is simply a plant component, in particular involving no specification. Moreover, the centralized/local scalable supervisors we design are distinct from the supervisor given in [21], because our centralized supervisor works effectively for the entire system and local supervisors for individual plant components.

The work most related to ours is reported in [22], [23]. Therein the same type of multi-agent systems is investigated and relabeling maps are used to generate template structures. Various properties of the relabeling map are proposed which characterize relations between the relabeled system and the original one. Moreover, a supervisor is designed that is provably independent of agent numbers, when these numbers exceed a certain threshold value. The design of the supervisor is, however, based on first computing the synchronous product of all agents, which can be computationally expensive. This can be relieved by using state tree structures [23], but the computation is still dependent on the agent numbers and thus the supervisor has to be recomputed or reconfigured whenever the number of agents changes. By contrast, our synthesis is based only on the template structures and thus independent of the agent numbers; furthermore the state size of our designed supervisor is always independent of the number of agents, with no threshold value required.

The rest of this paper is organized as follows. Section II introduces preliminaries and formulates the scalable supervisory control synthesis problem. Section III solves the problem by designing a scalable supervisor, and shows that the sufficient conditions for solving the problem are efficiently verifiable. Section IV designs scalable local controllers for individual agents, and Section V presents three examples to illustrate scalable supervisors and local controllers. Finally Section VI states our conclusions.

II. Preliminaries and Problem Formulation

A. Preliminaries

Let the DES plant to be controlled be modeled by a generator

\[ G = (Q, \Sigma, \delta, q_0, Q_m) \]

where \( \Sigma = \Sigma_c \cup \Sigma_u \) is a finite event set that is partitioned into a controllable event subset and an uncontrollable subset, \( Q \) is the finite state set, \( q_0 \in Q \) the initial state, \( Q_m \subseteq Q \) the set of marker states, and \( \delta : Q \times \Sigma \to Q \) the (partial) transition function. Extend \( \delta \) in the usual way such that \( \delta : Q \times \Sigma^* \to Q \). The closed behavior of \( G \) is the language

\[ L(G) := \{ s \in \Sigma^* | \delta(q_0, s)! \} \subseteq \Sigma^* \]

in which the notation \( \delta(q_0, s)! \) means that \( \delta(q_0, s) \) is defined. The marked behavior of \( G \) is

\[ L_m(G) := \{ s \in L(G) | \delta(q_0, s) \in Q_m \} \subseteq L(G) \]

A string \( s_1 \) is a prefix of another string \( s \), written \( s_1 \leq s \), if there exists \( s_2 \) such that \( s_1s_2 = s \). The prefix closure of \( L_m(G) \) is

\[ \overline{L_m(G)} := \{ s_1 \in \Sigma^* | (\exists s \in L_m(G))s_1 \leq s \} \]

We say that \( G \) is nonblocking if \( \overline{L_m(G)} = L(G) \).

A language \( K \subseteq L_m(G) \) is controllable with respect to \( L(G) \) provided \( K \Sigma_u \cap L(G) \subseteq \overline{K} \) [6]. Let \( E \subseteq L_m(G) \) be a specification language for \( G \), and define the set of all
sublanguages of $E$ that are controllable with respect to $L(G)$ by

$$C(E) := \{K \subseteq E \mid K \subseteq \overline{Kt} \cap L(G) \subseteq \overline{Kt}\}.$$ 

Then $C(E)$ has a unique supremal element [6]

$$\sup C(E) = \cup\{K \mid K \in C(E)\}.$$ 

For describing a modular structure of plant $G$, we first introduce a relabeling map. Let $T$ be a set of new events, i.e. $\Sigma \cap T = \emptyset$. Define a relabeling map $R : \Sigma \to T$ such that for every $\sigma \in \Sigma$, 

$$R(\sigma) = \tau, \quad \tau \in T.$$ 

In general $R$ is surjective but need not be injective.

For $\sigma \in \Sigma$, let $[\sigma]$ be the set of events in $\Sigma$ that have the same $R$-image as $\sigma$, i.e.

$$[\sigma] := \{\sigma' \in \Sigma \mid R(\sigma') = R(\sigma)\}.$$ 

Then $\Sigma = [\sigma_1] \cup [\sigma_2] \cup \ldots \cup [\sigma_k]$, for some $k \geq 1$, and $T$ can be written as $T = \{R(\sigma_1), R(\sigma_2), \ldots, R(\sigma_k)\}$. We require that $R$ preserve controllable/uncontrollable status of events in $\Sigma$; namely $R(\sigma)$ is a controllable event if and only if $\sigma \in \Sigma_c$. Thus $T_c := \{R(\sigma) \mid \sigma \in \Sigma_c\}$, $T_u := \{R(\sigma) \mid \sigma \in \Sigma_u\}$, and $T = T_c \cup T_u$.

We extend $R$ such that $R : \Sigma^* \to T^*$ according to

(i) $R(\varepsilon) = \varepsilon$, where $\varepsilon$ denotes the empty string;
(ii) $R(\sigma) = \tau$, $\sigma \in \Sigma$ and $\tau \in T$;
(iii) $R(\sigma\tau) = R(\sigma)R(\tau)$, $\sigma \in \Sigma$ and $\tau \in T$.

Note that $R(\sigma) \neq \varepsilon$ for all $\sigma \in \Sigma^* \setminus \{\varepsilon\}$.

Further extend $R$ for languages, i.e. $R : Pwr(\Sigma^*) \to Pwr(T^*)$, and define

$$R(L) = \{R(s) \mid s \in L\}, \quad L \subseteq \Sigma^*.$$ 

The inverse-image function $R^{-1}$ of $R$ is given by $R^{-1} : Pwr(T^*) \to Pwr(\Sigma^*)$:

$$R^{-1}(H) = \{s \in \Sigma^* \mid R(s) \in H\}, \quad H \subseteq T^*.$$ 

Note that $R^{-1}(H) = H$, $H \subseteq T^*$ while $R^{-1}(L) \supseteq L$, $L \subseteq \Sigma^*$. We say that $L \subseteq \Sigma^*$ is $(G, R)$-normal if $R^{-1}(L) \cap L(G) \subseteq L$; this property will turn out to be important in Section III below. Several useful properties of $R$ and $R^{-1}$ are presented in the following lemma, whose proof is given in Appendix.

**Lemma 1:** For $R : Pwr(\Sigma^*) \to Pwr(T^*)$ and $R^{-1} : Pwr(T^*) \to Pwr(\Sigma^*)$, the following statements are true.

(i) $R(\overline{L}) = \overline{R(L)}$, $L \subseteq \Sigma^*$;
(ii) $R(L_1 \cap L_2) \subseteq R(L_1) \cap R(L_2)$, $L_1, L_2 \subseteq \Sigma^*$;
(iii) $R^{-1}(H) = R^{-1}(H)$, $H \subseteq T^*$;
(iv) $R^{-1}(H_1 \cap H_2) = R^{-1}(H_1) \cap R^{-1}(H_2)$, $H_1, H_2 \subseteq T^*$.

We now discuss computation of $R$, $R^{-1}$ by generators. Let $R : \Sigma^* \to T^*$ be a relabeling map and $G = (Q, \Sigma, \delta, q_0, Q_m)$ a generator. First, relabel each transition of $G$ to obtain $G_T = (Q, T, \delta_T, q_0, Q_m)$, where $\delta_T : Q \times T \to Q$ is defined by

$$\delta_T(q_1, \tau) = q_2 \text{ if } (\exists \sigma \in \Sigma) R(\sigma) = \tau \text{ and } \delta(q_1, \sigma) = q_2.$$ 

Hence $L_m(G_T) = R(L_m(G))$ and $L(G_T) = R(L(G))$.

However, $G_T$ as given above may be nondeterministic [6]. Thus apply subset construction [6] to convert $G_T$ into a deterministic generator $H = (Z, T, \zeta, z_0, Z_m)$, with $L_m(H) = L_m(G_T)$ and $L(H) = L(G_T)$. See Fig. 1 for an illustrative example.

**Lemma 2:** If $G$ is nonblocking, then the relabeled generator $H$ is also nonblocking.

**Proof:** Suppose that $G$ is nonblocking, i.e. $L_m(G) = L(G)$. Then

$$\Rightarrow L_m(G) = L(G) \quad \text{(by Lemma 1(i))}$$

$$\Rightarrow L_m(H) = L(H)$$

namely $H$ is nonblocking. \hfill \Box

Conversely, to inverse-relabel $H$, simply replace each transition $\tau(\in T)$ of $H$ by those $\sigma(\in \Sigma)$ with $R(\sigma) = \tau$; thus one obtains $G' = (Z, \Sigma, \zeta', z_0, Z_m)$, where $\zeta' : Z \times \Sigma \to Z$ is defined by

$$\zeta'(z_1, \sigma) = z_2 \iff (3\tau(\in T)) R(\sigma) = \tau \text{ and } \zeta(z_1, \tau) = z_2.$$ 

It is easily verified that $L_m(G') = R^{-1}L_m(H)$ and $L(G') = R^{-1}L(H)$. Note that $G'$ as given above is deterministic (since $H$ is), and has the same number of states as $H$; namely inverse-relabeling does not change state numbers. Note that $L_m(G') \supseteq L_m(G)$ and $L(G') \supseteq L(G)$. Refer again to Fig. 1 for illustration. Henceforth we shall write $R(G) := H$ and $R^{-1}(H) := G'$.

**B. Problem Formulation**

Let $R : \Sigma^* \to T^*$ be a relabeling map, and $G = \{G_1, \ldots, G_k\}$ be a set of generators. We say that $G$ is a

\[1\] The worst-case complexity of subset construction is exponential. In the problem considered in this paper, nevertheless, the generators that need to be relabeled typically have small state sizes, and hence their relabeled models may be easily computed. This point will be illustrated by examples given below.
Design a scalable supervisor $\text{SSUP}$ (a nonblocking generator) such that

(i) The number of states of $\text{SSUP}$ and its computation are independent of the number of agents $n_i$, $i \in \{1, \ldots, l\}$;

(ii) $L_m(\text{SSUP}) \cap L_m(G)$ is nonempty and controllable with respect to $L(G)$, i.e. $\emptyset \neq L_m(\text{SSUP}) \cap L_m(G) \subseteq L_m(\text{SUP})$.

It would be ideal to have $L_m(\text{SSUP}) \cap L_m(G) = L_m(\text{SUP})$. Inasmuch as this requirement might be too strong to admit any solution to the problem, we shall consider (ii) above.

More generally, one may consider DES isomorphism (e.g. [14]) and say that $\hat{G} = \{G_1, \ldots, G_k\}$ is a similar set if $R(G_i)$ and $R(G_j)$ are isomorphic for all $i, j \in \{1, \ldots, k\}$. For simplicity of presentation we use the definition in [14], and subsequent development may be readily extended to the more general case using DES isomorphism.

More generally, one may consider DES isomorphism (e.g. [14]), and say that $\hat{G} = \{G_1, \ldots, G_k\}$ is a similar set if $R(G_i)$ and $R(G_j)$ are isomorphic for all $i, j \in \{1, \ldots, k\}$. For simplicity of presentation we use the definition in [14], and subsequent development may be readily extended to the more general case using DES isomorphism.

1. Design a scalable supervisor $\text{SSUP}$ (a nonblocking generator) such that

(i) The number of states of $\text{SSUP}$ and its computation are independent of the number of agents $n_i$, $i \in \{1, \ldots, l\}$;

(ii) $L_m(\text{SSUP}) \cap L_m(G)$ is nonempty and controllable with respect to $L(G)$, i.e. $\emptyset \neq L_m(\text{SSUP}) \cap L_m(G) \subseteq L_m(\text{SUP})$.

It would be ideal to have $L_m(\text{SSUP}) \cap L_m(G) = L_m(\text{SUP})$. Inasmuch as this requirement might be too strong to admit any solution to the problem, we shall consider (ii) above.

More generally, one may consider DES isomorphism (e.g. [14]) and say that $\hat{G} = \{G_1, \ldots, G_k\}$ is a similar set if $R(G_i)$ and $R(G_j)$ are isomorphic for all $i, j \in \{1, \ldots, k\}$. For simplicity of presentation we use the definition in [14], and subsequent development may be readily extended to the more general case using DES isomorphism.

More generally, one may consider DES isomorphism (e.g. [14]) and say that $\hat{G} = \{G_1, \ldots, G_k\}$ is a similar set if $R(G_i)$ and $R(G_j)$ are isomorphic for all $i, j \in \{1, \ldots, k\}$. For simplicity of presentation we use the definition in [14], and subsequent development may be readily extended to the more general case using DES isomorphism.

More generally, one may consider DES isomorphism (e.g. [14]) and say that $\hat{G} = \{G_1, \ldots, G_k\}$ is a similar set if $R(G_i)$ and $R(G_j)$ are isomorphic for all $i, j \in \{1, \ldots, k\}$. For simplicity of presentation we use the definition in [14], and subsequent development may be readily extended to the more general case using DES isomorphism.

More generally, one may consider DES isomorphism (e.g. [14]) and say that $\hat{G} = \{G_1, \ldots, G_k\}$ is a similar set if $R(G_i)$ and $R(G_j)$ are isomorphic for all $i, j \in \{1, \ldots, k\}$. For simplicity of presentation we use the definition in [14], and subsequent development may be readily extended to the more general case using DES isomorphism.

More generally, one may consider DES isomorphism (e.g. [14]) and say that $\hat{G} = \{G_1, \ldots, G_k\}$ is a similar set if $R(G_i)$ and $R(G_j)$ are isomorphic for all $i, j \in \{1, \ldots, k\}$. For simplicity of presentation we use the definition in [14], and subsequent development may be readily extended to the more general case using DES isomorphism.
III. SCALABLE SUPERVISORY CONTROL

In this section we design a scalable supervisor to solve the Scalable Supervisory Control Synthesis Problem (SSCSP), under certain sufficient conditions. Moreover we will show that these conditions are easily verified with low computational effort.

Consider the plant $G$ as described in Section II.B. Let $\Sigma(=\Sigma_c|\bigcup\Sigma_n)$ be the event set of $G$, and $R: \Sigma \rightarrow T$ a relabeling map. The procedure of designing a scalable supervisor is as follows, (P1)-(P4), which involves first synthesizing a supervisor for ‘relabeled system’ under $R$ and then inverse-relabeling the supervisor.

(P1) Compute $H$ as the synchronous product of the template generators $H_i$ in (2), i.e.

$$H := ||_{i \in \{1, \ldots, l}\} H_i.$$  

We call $H$ the relabeled plant under $R$; it is nonblocking by Assumptions (A1), (A2). The event set of $H$ is $T = T_c \bigcup T_u$, where $T_c = R(\Sigma_c)$ and $T_u = R(\Sigma_u)$. The number of states of $H$ is independent of $n_i$, the number of agents in each group $i \in \{1, \ldots, l\}$.

(P2) Compute $F := R(E)$, where $E \subseteq \Sigma^*$ is the specification imposed on $G$. We call $F \subseteq T^*$ the relabeled specification imposed on $H$.

(P3) Synthesize a relabeled supervisor $RSUP$ (a nonblocking generator) such that

$$L_m(RSUP) = \sup C(L_m(H) \cap F) \subseteq T^*.$$  

The number of states of $RSUP$ is independent of the number of agents, since $H$’s state size is so.

(P4) Inverse-relabel $RSUP$ to derive $SSUP$, i.e.

$$SSUP := R^{-1}(RSUP)$$

with the marked behavior

$$L_m(SSUP) = R^{-1}(L_m(RSUP)) \subseteq \Sigma^*.$$  

By the inverse-relabeling computation introduced in Section II.A, $SSUP$ computed in (3) has the same number of states as $RSUP$. It then follows that the state size of $SSUP$ is independent of the number of agents in plant $G$.

Moreover, it is easily observed that $SSUP$ is nonblocking (since $RSUP$ is), and its computation does not depend on the number $n_i$ of agents in each group $i \in \{1, \ldots, l\}$. The above design procedure is demonstrated with an example displayed in Fig. 3.

Our main result is the following.

Theorem 1: Consider the plant $G$ as described in Section II.B and suppose that Assumptions (A1), (A2), (A3) hold. If the specification $E \subseteq \Sigma^*$ is $(G, R)$-normal and $L_m(H)$ is controllable with respect to $R(L(G))$, then $SSUP$ in (3) is a scalable supervisor that solves SSCSP.

Note that the state size of $SSUP$ is related to the number of groups that the plant is divided into, as well as the state size of the generator representing the relabeled specification $F$. In this paper we focus on the scalability of the relabeling procedure with respect to the number of agents, and thus assume the above two factors fixed for each problem we consider. In applications where these factors may be relevant, different approaches will need to be developed.

Fig. 3. Consider the small factory example in Fig. 2, and a specification that protects the buffer (with two slots) against overflow and underflow. This specification is represented by $E$. In (P1), compute the relabeled plant $H = H_1|H_2$, where $H_1, H_2$ are from Fig. 2. In (P2), compute the relabeled specification $R(E)$. In (P3), compute the relabeled supervisor $RSUP$ for $H$ and $R(E)$. Finally in (P4), compute $R^{-1}(RSUP)$ to derive the scalable supervisor $SSUP$.

Theorem 1 provides two sufficient conditions under which $SSUP$ in (3) is a solution to SSCSP. The first condition is the $(G, R)$-normality of $E (\subseteq \Sigma^*)$, i.e. $R^{-1}(R(E)) \cap L_m(G) \subseteq E$. This requires that $E$ (in particular $E \cap L_m(G)$) can be recovered from the relabeling mapping along with a procedure for deciding membership in $L_m(G)$. This requirement on the specification $E$ should be of no surprise, as in (P2) the relabeled specification $F$ is not arbitrary but computed by relabeling $E$, and in (P4) the scalable supervisor $SSUP$ is derived by inverse-relabeling the relabeled supervisor that enforces $F$. This situation is similar to decentralized/hierarchical supervisory control with natural projections (e.g. [6, Sec. 6.8]).

The second condition is the controllability of $L_m(H)$ with respect to $R(L(G))$, i.e. $L_m(H)\Sigma_u \cap R(L(G)) \subseteq L_m(H)$. This means that the relabeled plant should be controllable with respect to the relabeling of the original plant $G$; in other words, the relabeling operation should not remove uncontrollable events that are allowed by $G$. As we shall see below, this condition is essential in proving the controllability of $L_m(SSUP) \cap L_m(G)$ with respect to $L(G)$.

For the success of our scalable supervisory control synthesis, it is important to be able to efficiently verify these two sufficient conditions. At the appearance, however, both conditions seem to require computing $G$ which would be computationally infeasible for large systems. Nevertheless, as will be shown in the subsection below, under Assumptions (A1), (A2) the second condition may be efficiently verified, while the first condition may be verified by a stronger but efficiently-checkable condition.

Thus under the two easily checkable sufficient conditions, Theorem 1 asserts that $SSUP$ in (3) is a valid scalable
supervisor whose state size is independent of the number of agents in the plant. The advantages of this scalability are, (i) computation of SSUP is independent of the number of agents and thus this method may handle systems with large numbers of agents; (ii) SSUP does not need to be recomputed or reconfigured if and when some agents are removed due to failure or added for increasing productivity.

For the example in Fig. [3], it is verified that both sufficient conditions of Theorem 1 are satisfied, and therefore the derived scalable supervisor SSUP is a solution to SSCSP.

To prove Theorem 1 we need to the following lemmas.

**Lemma 3:** Consider the plant G as described in Section II.B and suppose that Assumptions (A1), (A2) hold. Then H is nonblocking, and
\[ L_m(H) \subseteq R(L_m(G)). \]

**Lemma 4:** Consider the plant G as described in Section II.B and suppose that Assumptions (A1), (A2) hold. Then SSUP and G are nonconflicting, i.e.
\[ L_m(SSUP) \cap L_m(G) = L_m(SSUP) \cap L_m(G). \]

The proofs of the above Lemmas are referred to Appendix. Now we are ready to provide the proof of Theorem 1.

**Proof of Theorem 1** That the number of states of SSUP and its computation are independent of the number \( n_i \) of agents for all \( i \in \{1, \ldots, l\} \) has been asserted following (P4) of designing SSUP. Hence to prove that SSUP is a scalable supervisor that solves SSCSP, we will show that \( \emptyset \neq L_m(SSUP) \cap \Sigma_u \subseteq L_m(SUP) \cap L_m(G) \).

The fact that \( L_m(SSUP) \cap L_m(G) \) is nonempty follows immediately from Assumption (A3). Indeed, it is readily observed that when \( n_i = 1 \) for all \( i \in \{1, \ldots, l\} \), R is a bijective relabeling map, and thus SSUP designed by (P1)-(P4) is isomorphic to the monolithic supervisor SUP (with the isomorphism R).

It remains to show that \( L_m(SSUP) \cap L_m(G) \subseteq L_m(SUP) = \text{sup}C(E \cap L_m(G)). \) For this we will prove that (i) \( L_m(SSUP) \cap L_m(G) \) is controllable with respect to \( L(G) \), and (ii) \( L_m(SSUP) \cap L_m(G) \subseteq E \cap L_m(G) \). For (i) let \( s \in L_m(SSUP) \cap L_m(G) \), \( \sigma \in \Sigma_u \), \( s \sigma \in L(G) \). Then
\[ s \in L_m(SSUP) \cap L_m(G) \]
\[ \Rightarrow (\exists t)st \in L_m(SSUP) \]
\[ \Rightarrow st \in R^{-1}L_m(RSUP) \] (by (P4))
\[ \Rightarrow R(st) \in L_m(RSUP) \subseteq L_m(H) \]
\[ \Rightarrow R(s) \in L_m(RSUP) \cap L_m(H). \]

Since \( s \sigma \in L(G) \), we have \( R(s)R(\sigma) \in R(L(G)) \) where \( R(\sigma) \in T_u \) (since \( \sigma \in \Sigma_u \)). It then follows from the controllability of \( L_m(H) \) with respect to \( R(L(G)) \) that \( R(s)R(\sigma) \in L_m(H) \) (\( H \) is nonblocking by Lemma 3). Now use the controllability of \( L_m(RSUP) \) with respect to \( L(H) \) to derive \( R(s)R(\sigma) \in L_m(RSUP) \), and in turn
\[ s \sigma \in R^{-1}L_m(RSUP) \subseteq R^{-1}L_m(RSUP) \]
\[ \Rightarrow s \sigma \in R^{-1}L_m(RSUP) = L_m(SSUP). \]

In the derivation above, we have used Lemma 3(iii). In addition, since \( s \sigma \in L(G) = L_m(G) \) (\( G \) is nonblocking by Assumption (A1)), we have
\[ s \sigma \in L_m(SSUP) \cap L_m(G) \]

Under Assumptions (A1), (A2), it follows from Lemma 4 that SSUP and G are nonconflicting, i.e.
\[ L_m(SSUP) \cap L_m(G) = L_m(SSUP) \cap L_m(G). \] Hence
\[ s \sigma \in L_m(SSUP) \cap L_m(G), \] which proves (i).

For (ii) let \( s \in L_m(SSUP) \cap L_m(G) \). Then
\[ s \in R^{-1}L_m(RSUP) \cap L_m(G) \]
\[ \Rightarrow s \in L_m(G) \] & \( R(s) \in L_m(RSUP) \subseteq F = R(E) \]
\[ \Rightarrow s \in L_m(G) \] & \( s \in R^{-1}R(s) \subseteq R^{-1}R(E). \]

Since E is (G, R)-normal, i.e. \( R^{-1}R(E) \subseteq E \), we derive \( s \in E \cap L_m(G) \), which proves (ii). The proof is now complete.

**Proposition 2** Consider the plant G as described in Section II.B and suppose that Assumptions (A1), (A2) hold. For each group \( i \in \{1, \ldots, l\} \) if \( L_m(H_i) \) is controllable with respect to \( R(L(G_{i1} \cap G_{i2})) \), then \( L_m(H) \) is controllable with respect to \( R(L(G)) \).

Proposition 2 asserts that the controllability of \( L_m(H) \) with respect to \( R(L(G)) \) may be checked in a modular fashion: namely it is sufficient to check the controllability of \( L_m(H_i) \) for each group with respect to only two component agents. As a result, the computational effort of checking the condition is low.
Consider a group of 2 machines $G_{11}, G_{12}$. Let $\Sigma = \Sigma_1 \cup \Sigma_2$, where $\Sigma_1 = \{11, 21\}$ and $\Sigma_2 = \{10, 20\}$. Let $T = \{0, 1\}$, and the relabeling map $R : \Sigma \to T$ with $R(11) = (21) = 1 \in T_1$, $R(21) = R(20) = 0 \in T_2$. Under $R$, $H_1 = R(G_{11}) = R(G_{12})$ and $R(G_{11} \| G_{12})$ are displayed. Observe that $L_m(H_1)$ is not controllable with respect to $R(L(G_{11} \| G_{12}))$: let $t = 0 \in L_m(H_1)$ and $\tau = 0 \in T_2$ such that $\tau \in R(L(G_{11} \| G_{12}))$, but $\tau \notin L_m(H_1)$.

Note that the condition in Proposition 2.2 $L_m(H)$ being controllable with respect to $R(L(G_{11} \| G_{12}))$ does not always hold. An example where this condition fails is shown in Fig. 4.

To prove Proposition 2.2 we need the following two lemmas. For convenience it is assumed that Assumptions (A1), (A2) hold henceforth in this subsection.

**Lemma 5:** Let $i \in \{1, \ldots, l\}$. If $L_m(H_i)$ is controllable with respect to $R(L(G_{1i} \| G_{2i}))$, then $L_m(H_i)$ is controllable with respect to $R(L(G_{1i}, G_{2i} \| G_{3i}))$.

**Proof:** Let $i \in \{1, \ldots, l\}$, $t \in \mathbb{Z}^+ = R(L(G_{1i} \| G_{2i}))$, $\tau \in T_2$, and $\tau \in R(L(G_{1i}, G_{2i} \| G_{3i}))$. We shall show that $\tau \in R(L(G_{1i} \| G_{2i})) = L_m(H_i)$. By $t \in R(L(G_{1i} \| G_{2i}))$ we derive

$$t \in R(L(G_{1i} \| G_{2i})) = R(L(G_{1i})).$$

By Assumption (A1), $G_{1i}, G_{2i}, G_{3i}$ do not share events, the string $s$ must be only in $G_{1i}$. Thus for each $\sigma \in \Sigma_1$ with $R(\sigma) = \tau$, if $\sigma \in L(G_{1i} \| G_{2i} \| G_{3i})$, then

- either $\sigma \in L(G_{1i})$ if $\sigma$ is an event of $G_{1i}$
- or $\sigma \in L(G_{1i} \| G_{2i})$ if $\sigma$ is an event of $G_{2i}$
- or $\sigma \in L(G_{1i} \| G_{3i})$ if $\sigma$ is an event of $G_{3i}$.

Hence $\tau = R(\sigma) \in R(L(G_{1i} \| G_{2i}) \| G_{3i})$ implies

- either $R(\sigma) \in R(L(G_{1i}))$
- or $R(\sigma) \in R(L(G_{1i} \| G_{2i}))$
- or $R(\sigma) \in R(L(G_{1i} \| G_{3i})) = R(L(G_{1i} \| G_{2i})).$

For the latter two cases, use the controllability of $L_m(H_i)$ with respect to $R(L(G_{1i} \| G_{2i}))$ to derive $R(\sigma) \in R(L(G_{1i}))$. Therefore, after all, $\tau \in R(L(G_{1i})) = R(L_m(G_{1i}))$ as required.

Applying Lemma 5 inductively, one derives that if $L_m(H_i)$ ($i \in \{1, \ldots, l\}$) is controllable with respect to $R(L(G_{1i} \| G_{2i}))$, then $L_m(H_i)$ is controllable with respect to $R(L(G_{1i}) \| J \in \{1, \ldots, \}$, where $G_i = ||_{j \in \{1, \ldots, \} G_{ij}}.$

**Lemma 6:** If, for every $i \in \{1, \ldots, l\}$, $L_m(G_i)$ is controllable with respect to $R(L(G_i))$, then $L_m(H) = L_m(\bigcup_{i \in \{1, \ldots, \} H_i)$ is controllable with respect to $R(L(G))$.

This is a known result given Assumption (A2); see e.g. [12], [24]. We are now ready to prove the present of Proposition 2.2.

**Proof of Proposition 2.2** Combining Lemmas 2 and 2.5 it directly follows that if $L_m(H_i)$ ($i \in \{1, \ldots, l\}$) is controllable with respect to $R(L(G_{1i} \| G_{2i}))$, then $L_m(H)$ is controllable with respect to $R(L(G))$. \(\square\)

IV. SCALABLE DISTRIBUTED CONTROL

So far we have synthesized a scalable supervisor $SSUP$ that effectively controls the entire multi-agent system, i.e. $SSUP$ is a centralized controller. For the type of system considered in this paper which consists of many independent agents, however, it is also natural to design a distributed control architecture where each individual agent acquires its own local controller (thereby becoming autonomous).

Generally speaking, a distributed control architecture is advantageous in reducing (global) communication load, since local controllers typically need to interact only with their (nearest) neighbors. A distributed architecture might also be more fault-tolerant, as partial failure of local controllers or the corresponding agents would unlikely to overtake the whole system.

For these potential benefits, we aim in this section to design for the multi-agent system a distributed control architecture. In particular, we aim to design local controllers that have the same scalability as the centralized $SSUP$; namely, their state sizes and computation are independent of the number of agents in the system. Thus when some agents break down and/or new agents are added in, there is no need of recomputing or reconfiguring these local controllers.

Let us now formulate the following Scalable Distributed Control Synthesis Problem (SDCSP):

**Design a set of scalable local controllers $SLOC_{ij}$ (a non-blocking generator), one for each agent $G_{ij}$ ($i \in \{1, \ldots, l\}, j \in \{1, \ldots, n_i\}$) such that:**

1. The number of states and computation of $SLOC_{ij}$ are independent of the number $n_i$ of agents for all $i \in \{1, \ldots, l\}$
2. The set of $SLOC_{ij}$ is (collectively) control equivalent to the scalable supervisor $SSUP$ with respect to plant $G$, i.e.

$$\left( \bigcap_{i \in \{1, \ldots, l\}} \bigcap_{j \in \{1, \ldots, n_i\}} L_m(SLOC_{ij}) \right) \cap L_m(G) = L_m(SSUP) \cap L_m(G).$$

(4)

To solve SDCSP, we employ a known technique called supervisor localization [14], [15], [16], which works to decompose an arbitrary supervisor into a set of local controllers whose collective behavior is equivalent to that supervisor. Since we have synthesized $SSUP$, the scalable supervisor, a straightforward approach would be to apply supervisor
localization to decompose the associated controlled behavior $L_m(SSUP) \cap L_m(G)^6$]. This approach would require, however, the computation of $G$ which is infeasible for large systems and cause the resulting local controllers non-scalable.

Instead we propose the following procedure for designing scalable local controllers $SLOC_{ij}$, for $i \in \{1, \ldots, l\}$ and $j \in \{1, \ldots, n_i\}$.

(Q1) Apply supervisor localization to decompose the relabeled supervisor $RSUP$ into relabeled local controllers $RLOC_i$, $i \in \{1, \ldots, l\}$, such that [16]

\[
\left( \bigcap_{i \in \{1, \ldots, l\}} L_m(RLOC_i) \right) \cap L_m(H) = L_m(RSUP).
\]

(Q2) Compute $\text{trim}(RLOC_i[H_i])$, where $\text{trim}(\cdot)$ operation removes blocking states (if any) of the argument generator.

(Q3) Inverse-relabel $\text{trim}(RLOC_i[H_i])$ to obtain $SLOC_{ij}$ ($j \in \{1, \ldots, n_i\}$), i.e.

\[
SLOC_{ij} := R^{-1}(\text{trim}((RLOC_i[H_i]))) \tag{5}
\]

Notice that the computations involved in the above procedure are independent of the number $n_i$ ($i \in \{1, \ldots, l\}$) of agents. In (Q1), computing $RLOC_i$ by localization requires computing $RSUP$ and $H$ (in (P1) and (P3) respectively), both of which are independent of $n_i$. In (Q2), for the synchronous product both $RLOC_i$ and $H_i$ are independent of $n_i$, while trim may only reduce some states. Finally in (Q3), inverse-relabeling does not change the number of states. Therefore the state number of the resulting scalable local controller $SLOC_{ij}$ and its computation are independent of the number $n_i$ ($i \in \{1, \ldots, l\}$) of agents.

The synchronous product in (Q2) is indeed crucial to ensure the correctness of the resulting local controllers. If we did not compute this synchronous product and set the local controllers to be $R^{-1}(\text{trim}(RLOC_i))$, then such local controllers cannot even guarantee that the controlled behavior satisfies the imposed specification, as will be demonstrated in Section V.A below.

On the other hand, the synchronous product in (Q2) may produce blocking states; such an example is provided in Section V.B. Thus the trim operation is needed to ensure that the resulting $SLOC_{ij}$ is a nonblocking generator.

In addition, note that $SLOC_{ij}$ are the same for all $j \in \{1, \ldots, n_i\}$. This means that every agent $G_{ij}$ in the same group $G_i$ obtains the same local controller, although each local controller will be dedicated to enabling/disabling only the controllable events originated from its associated agent.

The main result of this section is the following.

**Theorem 2:** The set of $SLOC_{ij}$ ($i \in \{1, \ldots, l\}$, $j \in \{1, \ldots, n_i\}$) as in (5) is a set of scalable local controllers that solves SDCSP.

**Proof:** That the number of states of $SLOC_{ij}$ and its computation are independent of the number $n_i$ of agents for all $i \in \{1, \ldots, l\}$, $j \in \{1, \ldots, n_i\}$ has been asserted following (Q3) of designing $SLOC_{ij}$. Hence to prove that the set of $SLOC_{ij}$ is a set of scalable local controllers that solves SDCSP, we will show [4].

From (Q1) we have

\[
\left( \bigcap_{i \in \{1, \ldots, l\}} L_m(RLOC_i) \right) \cap L_m(H) = L_m(RSUP)
\]

\[
\Rightarrow \left( \bigcap_{i \in \{1, \ldots, l\}} L_m(RLOC_i) \right) \cap \left( \bigcap_{i \in \{1, \ldots, l\}} L_m(H_i) \right) = L_m(RSUP)
\]

\[
\Rightarrow \bigcap_{i \in \{1, \ldots, l\}} \left( L_m(RLOC_i) \cap L_m(H_i) \right) = L_m(RSUP)
\]

\[
\Rightarrow \bigcap_{i \in \{1, \ldots, l\}} \left( L_m(\text{trim}(RLOC_i[H_i])) \right) = L_m(RSUP)
\]

Inverse-relabeling both sides and applying Lemma [4]iv), we derive

\[
R^{-1}\left( \bigcap_{i \in \{1, \ldots, l\}} L_m(\text{trim}(RLOC_i[H_i])) \right) = R^{-1}(L_m(RSUP))
\]

\[
\Rightarrow \bigcap_{i \in \{1, \ldots, l\}} R^{-1}(L_m(\text{trim}(RLOC_i[H_i]))) = R^{-1}(L_m(RSUP))
\]

\[
\Rightarrow \bigcap_{i \in \{1, \ldots, l\}} L_m(R^{-1}(\text{trim}(RLOC_i[H_i]))) = R^{-1}(L_m(RSUP))
\]

Finally it follows from (5) and (P4) that

\[
\left( \bigcap_{i \in \{1, \ldots, l\}} L_m(SLOC_{ij}) \right) = L_m(SSUP)
\]

\[
\Rightarrow \bigcap_{i \in \{1, \ldots, l\}, j \in \{1, \ldots, n_i\}} L_m(SLOC_{ij}) \cap L_m(G) = L_m(SSUP) \cap L_m(G).
\]

That is, [4] is established. \qed

V. ILLUSTRATING EXAMPLES

In this section, we provide three examples to illustrate our proposed scalable supervisory synthesis as well as distributed control. The first example is the extension of the small factory example (studied in Figs. 2, 3) to arbitrary numbers of input and output machines. The second example is a transfer line system, where we illustrate how to deal with more than one specification. The last example is called mutual exclusion, where the plant naturally contains only one group of agents; we demonstrate how to fit this type of multi-agent systems into our setting and apply our method to derive scalable supervisors and local controllers.

A. Small Factory

This example has already been presented in Figs. 2 and 3 with 2 input machines and 2 output machines. Here we consider the general case where there are $n$ input machines and $m$ output machines, for arbitrary $n,m \geq 1$. With the relabeling map $R$ in Fig. 2 extended in the obvious way, the agents are divided into 2 groups:
The template generators \( H_i \) \((i = 1, 2)\) are the same as those in Fig. 2 and the relabeled specification generator \( R(E) \) same as that in Fig. 3. As a result, the relabeled supervisor \( RSUP \) remains identical to that in Fig. 4 and the scalable supervisor \( SSUP \) obtained by inverse-relabeling \( RSUP \) is displayed in Fig. 5. The state size of \( SSUP \) and its computation are independent of the agent numbers \( n, m \).

It is verified that the two sufficient conditions of Theorem 1 are satisfied; in particular, the specification \( E \) in Fig. 3 satisfies \( E = R^{-1}(R(E)) \). Hence \( SSUP \) is a solution to SSCSP. We note, however, that the controlled behavior of \( SSUP \) is strictly smaller than the monolithic supervisor \( SUP \), i.e. \( L_m(SUP) \cap L_m(G) \subseteq L_m(SSUP) \), for any particular values of \( n, m \). The reason is as follows. By relabeling, each group is treated as if there were only one machine; this in effect disallows parallel operations of multiple machines in the same group. This means that for the input (resp. output) machine group, if one machine starts to work, other machines cannot start before the machine in operation finishes its work and returns to idle. For both the input machine group and the output machine group, this constraint is not enforced by \( SUP \) and therefore the controlled behavior of the scalable supervisor \( SSUP \) is more restrictive. The analysis above nevertheless provides a hint as to how scalable supervisors may be designed to achieve more permissive controlled behavior, or even equivalent behavior with the monolithic supervisor, which we shall study in our future work.

**Scalable distributed control.** Following the procedure (Q1)-(Q3) in Section IV, we compute the scalable local controllers for the individual agents. Specifically, as displayed in Fig. 6, \( SLOC_{1i} \) (5 states) is for the input machine \( G_{1i} \), \( i \in \{1, \ldots, n\} \); while \( SLOC_{2j} \) (6 states) is for the output machine \( G_{2j} \), \( j \in \{1, \ldots, m\} \). It is verified that the desired control equivalence between the set of local controllers and the supervisor \( SSUP \) in Fig. 5 is satisfied, i.e. \( 4 \) holds.

The control logic of the scalable local controllers is as follows. First for \( SLOC_{1i} \) \((i \in \{1, \ldots, n\})\), which controls only the event \( 1i1 \) of the input machine \( G_{1i} \), observe that event \( 1i1 \) is disabled at states 1, 3, and 4. Since buffer has two slots, disabling \( 1i1 \) at states 3, 4 is to protect buffer against overflow. On the other hand, disabling \( 1i1 \) at state 1 is not relevant to enforcing specification but rather the restriction brought in by relabeling; as we already mentioned above, relabeling disallows parallel operations of multiple machines in the same group.

Next for \( SLOC_{2j} \) \((j \in \{1, \ldots, m\})\), which is responsible only for event \( 2j1 \) of the output machine \( G_{2j} \), observe that event \( 2j1 \) is disabled at states 0, 2, 3, and 4. By inspection, disabling \( 2j1 \) at states 0, 2 is to protect buffer against underflow, while at states 3, 4 is due to the restriction of relabeling (i.e. no parallel operations of multiple output machines).

It is worth noting that the synchronous product operation in (Q2) is crucial. As displayed in Fig. 7 if we did not compute the synchronous product and instead used \( R^{-1}(RLOC_1) \) and \( R^{-1}(RLOC_2) \) as local controllers, where \( RLOC_1, RLOC_2 \) are the relabeled local controllers computed in (Q1), then the resulting controlled behavior would...
The system is divided into 3 groups. The agents are shown in Fig. 8. Based on their different roles, the test unit SSUP.

Let the relabeling map be given by

$$R(111) = 11, \quad R(112) = 12, \quad i \in \{1, \ldots, n\}$$

$$R(2j1) = 21, \quad R(2j2) = 22, \quad j \in \{1, \ldots, m\}$$

$$R(3l1) = 31, \quad R(3l2) = 32, \quad l \in \{1, \ldots, k\}$$

where odd-number events are controllable and even-number events are uncontrollable. It is easily observed that Assumptions (A1), (A2) hold.

The specification is to avoid underflow and overflow of buffers B1 and B2, which is enforced by the two generators E1 and E2 in Fig. 9. Thus the overall specification $E = E_1 \cap E_2$. One may easily verify that $R^{-1}R(L(G(E1))) = L_m(E1)$ and $R^{-1}R(L(G(E2))) = L_m(E2)$ hold. This is in fact sufficient to ensure $R^{-1}R(E) = E$ because

$$R^{-1}R(E) = R^{-1}(L_m(E_{1}) \cap L_m(E_{2}))$$

$$\subseteq R^{-1}(R(L_m(E_1)) \cap R(L_m(E_2)))$$

$$= R^{-1}(R(L_m(E_1))) \cap R^{-1}(R(L_m(E_2)))$$

$$= L_m(E_1) \cap L_m(E_2)$$

$$= E$$

where we use Lemma (iii) and (iv). The other direction $E \subseteq R^{-1}R(E)$ holds always. Thus the condition $R^{-1}R(E) = E$ may be checked in a modular fashion when $E$ is composed from multiple independent specifications. By Proposition 1 we have that $E$ is $(G, R)$-normal is verified. In addition, it is checked that $L_m(H_i) := L_m(R(G_{1i}))$ is controllable with respect to $R(L(G_{1i} \mid G_{2i}))$. By Proposition 2, we have that $L_m(H_i)$ is controllable with respect to $R(L(G))$. Therefore the two sufficient conditions of Theorem 1 are satisfied.

By the procedure (P1)-(P4), we design a scalable supervisor SSUP, displayed in Fig. 9. The state size of SSUP and its computation are independent of the agent numbers $n, m, k$. Moreover, the controlled behavior of SSUP is in fact equivalent to that of the monolithic supervisor SUP, i.e., $L_m(SSUP) \cap L_m(G) = L_m(SUP)$, for arbitrary fixed values of $n, m, k$. This is owing to that both buffers have only one slot, and thus the restriction due to relabeling is already enforced by the monolithic supervisor in order to satisfy the specification.

Scalable distributed control. Following the procedure (Q1)-(Q3) in Section IV, we compute the scalable local controllers for the individual agents. In (Q2), certain synchronous products turn out to be blocking, as displayed in Fig. 10 (upper part). Hence the trim operation in (Q2) is important to ensure the resulting local controllers are nonblocking. In Fig. 10 (lower part), $SLOC_{1i}$ (3 states) is for the machine $G_{1i}, i \in \{1, \ldots, n\}$; $SLOC_{2j}$ (3 states) for the machine $G_{2j}, j \in \{1, \ldots, m\}$; and $SLOC_{3l}$ (3 states) for the test unit $G_{3l}, l \in \{1, \ldots, k\}$. It is verified that the desired control equivalence between the set of local controllers and the supervisor SSUP in Fig. 9 is satisfied, i.e., (G) holds.
The control logic of the scalable local controllers is as follows. First for SLOC\(_i\) \((i \in \{1, \ldots, n\})\), which controls only the event 1\(i\) of machine G\(_{1i}\), observe that event 1\(i\) is disabled at states 1 and 2. This is to protect buffer B\(_1\) against overflow, as well as to ensure that there is no more than one workpiece in the ‘material-feedback’ loop of the transfer line.

Next for SLOC\(_{2j}\) \((j \in \{1, \ldots, m\})\), which is responsible only for event 2\(j\) of machine G\(_{2j}\), observe that event 2\(j\) is disabled at states 0 and 2. This is to protect buffer B\(_1\) against underflow and buffer B\(_2\) against overflow.

Finally for SLOC\(_{3l}\) \((l \in \{1, \ldots, k\})\), which is responsible only for event 3\(l\) of test unit G\(_{3l}\), observe that event 3\(l\) is disabled at states 0 and 2. This is to protect buffer B\(_2\) against underflow and buffer B\(_1\) against overflow.

C. Mutual Exclusion

In this last example, mutual exclusion, we demonstrate how to transform the problem into our setup and apply our scalable supervisory synthesis. There are \(n>1\) agents that compete to use a single resource; the specification is to prevent the resource being simultaneously used by more than one agent.

For this problem, it is natural to treat all agents as just one group. However, our approach would then relabel every agent to a single template model, to which the mutual exclusion specification could not be imposed (mutual exclusion specifies requirement between different agents). Thus in order to apply our synthesis method, we (artificially) separate the agents into two groups, with \(m\) and \(k\) agents respectively, such that \(n = m + k\). Namely

\[
\begin{align*}
G_1 &= \{G_{11}, \ldots, G_{1m}\} \\
G_2 &= \{G_{21}, \ldots, G_{2k}\}.
\end{align*}
\]

The generators of the agents separated into two groups and the specification are displayed in Fig. 10.

Let the relabeling map \(R\) be given by

\[
R(1i) = 11, \quad R(12) = 12, \quad i \in \{1, \ldots, m\} \\
R(2j) = 21, \quad R(2j) = 22, \quad j \in \{1, \ldots, k\}
\]

where odd-number events are controllable and even-number events are uncontrollable. It is readily checked that Assumptions (A1), (A2) hold. Moreover, it is verified that \(H_1 := L_m(R(G_{1i})) \quad (i = 1, 2)\) is controllable with respect to \(R(L(G_{1i}||G_{2j}))\), and \(R^{-1}R(E) = E;\) hence (by Propositions [1] and [2]) the sufficient conditions of Theorem 1 are satisfied.

By the procedure (P1)-(P4), we design a scalable supervisor SSUP, displayed in Fig. 12. Note that SSUP is identical to the specification \(E\), and the state size of SSUP and its computation are independent of the agent numbers \(m, k\) (hence the total number \(n\)). Moreover, the controlled behavior of SSUP is equivalent to that of the monolithic supervisor SUP, i.e., \(L_m(\text{SSUP}) \cap L_m(\text{G}) = L_m(\text{SUP})\), for any fixed value of \(n\). This is because there is only a single resource, and no matter how many agents are in the system, the resource can be used by only one agent at any given time. Thus the restriction due to relabeling has already been imposed by the mutual exclusion specification and enforced by the monolithic supervisor SUP.

Scalable distributed control. Following the procedure (Q1)-(Q3) in Section IV, we compute the scalable local controllers for the individual agents. Specifically, as displayed in Fig. 13 SLOC\(_{1i}\) (4 states) is for the first-group agent G\(_{1i}\), \(i \in \{1, \ldots, m\}\); while SLOC\(_{2j}\) (4 states) is for the second-group agent G\(_{2j}\), \(j \in \{1, \ldots, k\}\). It is verified that the desired
control equivalence between the set of local controllers and the supervisor \( SSUP \) in Fig. 12 is satisfied, i.e. (4) holds.

The control logic of the scalable local controllers is as follows. First for \( SLOC_{1i} \) \( (i \in \{1, \ldots, m\}) \), which controls only the event \( 1i1 \) of the first-group agent \( G_{1i1} \), observe that event \( 1i1 \) is disabled at states 1, 2, and 3. At all these states, the resource is being used by some agent; hence by mutual exclusion event \( 1i1 \) must be disabled.

It is worth noting that if the sequence \( 1i1,2j1 \) \( (j \in \{1, \ldots, k\}) \) occurred, which is allowed by \( SLOC_{1i} \), the mutual exclusion specification would be violated. Indeed \( 2j1 \) must be disabled after the occurrence of \( 1i1 \). However, since the local controller \( SLOC_{1i} \) is responsible only for event \( 1i1 \), the correct disablement of \( 2j1 \) \( (j \in \{1, \ldots, k\}) \) is left for another dedicated local controller \( SLOC_{2j} \). As we can see in \( SLOC_{2j} \), event \( 2j1 \) is disabled at states 1, 2, and 3. In particular, at state 1 (i.e. after \( 1i1 \) occurs) event \( 2j1 \) is correctly disabled to guarantee mutual exclusion (as expected). Therefore, while each local controller enables/disables only its locally-owned events, together they achieve correct global controlled behavior.

VI. CONCLUSIONS

We have studied multi-agent discrete-event systems that can be divided into several groups of independent and similar agents. We have employed a relabeling map to generate template structures, based on which scalable supervisors are designed whose state sizes and computational process are independent of the number of agents. We have presented two sufficient conditions for the validity of the designed scalable supervisors, and shown that these conditions may be verified with low computational effort. Moreover, on the scalable supervisor we have designed scalable local controllers, one for each component agent. Three examples have been provided to illustrate our proposed synthesis methods.

In future research, we aim to find conditions under which scalable supervisors may be designed to achieve controlled behavior identical to the monolithic supervisor. We also aim to search for new designs of scalable supervisors when the sufficient conditions of Theorem 1 fail to hold. Additionally we are interested in investigating, in the context of scalable supervisory control, the issue of partial observation.

REFERENCES

[1] Elmaraghy, H. (2005). Flexible and reconfigurable manufacturing systems paradigms, Int. J. Flexible Manufacturing Systems, vol. 17, no. 4, pp. 261-276.

[2] Wu, N. and Zhou, M. (2007). Deadlock resolution in automated manufacturing systems with robots, IEEE Trans. Automation Science and Engineering, vol. 4, no. 3, pp. 474-480.

[3] Wurman, P., D’Andrea, R., and Mountz, M. (2008). Coordinating hundreds of cooperative, autonomous vehicles in warehouses, AI Magazine, vol. 29, no. 1, pp. 9-19.

[4] Ramadge, P. J. and Wonham, W. M. (1987). Supervisory control of a class of discrete event processes, SIAM J. Control Optim., vol. 25, no. 1, pp. 206-230.

[5] Wonham, W. M. and Ramadge, P. J. (1987). On the supremal controllable sublanguage of a given language, SIAM J. Control Optim., vol. 25, no. 3, pp. 637-659.

[6] Wonham, W. M. (2016). Supervisory Control of Discrete-event Systems. Syst. Control Group, ECE Dept, Univ. Toronto, Toronto, ON, Canada.

[7] Gohari, P. and Wonham, W. M. (2000). On the complexity of supervisory control design in the RW framework, IEEE Trans. Syst., Man, Cybern., vol. 30, no. 5, pp. 643-652.

[8] Kumar, R. and Takai, S. (2012). A framework for control-reconfiguration following fault-detection in discrete event systems, in Proc. 8th IFAC Symp. Fault Detection, Supervision and Safety of Technical Processes, pp. 848-853.

[9] Nooruldeen, A. and Schmidt, K. (2015). State attraction under language specification for the reconfiguration of discrete event systems, EEE Trans. Autom. Control, vol. 60, no. 6, pp. 1630-1634.

[10] Wong, K. C. and Wonham, W. M. (1996). Hierarchical control of discrete-event systems. Discrete Event Dyna. Syst.: Theory Appl., vol. 6, no. 3, pp. 241-273.

[11] Wong, K. C. and Lee, S. (2002). Structural decentralized control of concurrent discrete-event systems, Eur. J. Control, vol. 8, pp. 477-491.

[12] Feng, L. and Wonham, W. M. (2008). Supervisory control architecture for discrete-event systems, IEEE Trans. Autom. Control, vol. 53, no. 6, pp. 1449-1461.

[13] Schmidt, K., Moor, T., and Perk, S. (2008). Nonblocking hierarchical control of decentralized discrete event systems, IEEE Trans. Autom. Control, vol. 53, no. 10, pp. 2252-2265.

[14] Cai, K. and Wonham, W. M. (2010). Supervisor localization: a top-down approach to distributed control of discrete-event systems. IEEE Trans. Autom. Control, vol. 55, no. 3, pp. 605-618.

[15] Cai, K. and Wonham, W. M. (2015). New results on supervisor localization, with case studies, Discrete Event Dynamic Systems, vol. 25, no. 1-2, pp. 203-226.

[16] Cai, K. and Wonham, W. M. (2016). Supervisor localization: a top-down approach to distributed control of discrete-event systems. IEEE Trans. Autom. Control, vol. 61, no. 1, pp. 491.

[17] Eyzell, J. M. and Cury, J. E. R. (2001). Exploiting symmetry in the synthesis of supervisors for discrete event systems. IEEE Trans. on Automatic Control, vol. 46, no. 9, pp. 1500-1505.

[18] Rohloff, K. and Lafortune, S. (2006). The verification and control of interacting similar discrete event systems. SIAM J. on Control Optim., vol. 45, no. 2, pp. 634-667.

[19] Su, R. (2013). Discrete-event modeling of multi-agent systems with broadcasting-based parallel composition, Automatica, vol. 49, no. 11, pp. 3502-3506.

[20] Su, R. and Lin, L. (2013). Synthesis of control protocols for multi-agent systems with similar actions. Proc. 52nd IEEE Conf. Decision Control, pp. 6986-6991.

[21] Su R., Lennartsson B. (2017). Control protocol synthesis for multi-agent systems with similar actions instantiated from agent and requirement templates. Automatica, 79:244-255.
By Assumption (A1) each plant component $R_H$ is nonblocking. Thus by Lemma 3(i) and Lemma 4 each $H_i$ is also nonblocking. Therefore by Assumption (A2) that $H_i$ do not share events, we derive that $H$ computed as the synchronous product of $H_i$ is nonblocking.

Next, we prove $L_m(H) \subseteq R(L_m(G))$. From (P1) we have

$$L_m(H) = \bigcup_{i \in \{1, \ldots, l\}} L_m(H_i)$$

$$= \bigcup_{i \in \{1, \ldots, l\}} R(L_m(G_{i}))$$

(by (2))

$$\subseteq \bigcup_{i \in \{1, \ldots, l\}} (R(L_m(G_{i1})) \cdots R(L_m(G_{in_i})))$$

$$= R \left( \bigcup_{i \in \{1, \ldots, l\}} (L_m(G_{i1}) \cdots L_m(G_{in_i})) \right)$$

(by Assumptions (A1), (A2))

$$= R(L_m(G)).$$

The proof is now complete. □

Finally we provide the proof of Lemma 4. This proof in fact has been given in the full version of [22], which is currently under review and there is no online version we can refer to. For completeness (for the review of this paper), we reproduce the proof here.

**Proof of Lemma 4** (⊆) This direction is always true.

(2) Let $s \in L_m(\Sigma_{sup}) \cap L_m(G)$. Then

$$s \in L_m(\Sigma_{sup})$$

$$\Rightarrow s \in R^{-1}(L_m(\Sigma_{sup})) \quad \text{(by (P4))}$$

$$\Rightarrow s \in R^{-1}(L_m(\Sigma_{sup})) \quad \text{(by Lemma 4(iii))}$$

$$\Rightarrow R(s) \in L_m(\Sigma_{sup}).$$

Let $t := R(s)$. Then there exists $w \in T^*$ such that $tw \in L_m(\Sigma_{sup})$. By (P3) we have $L_m(\Sigma_{sup}) \subseteq L_m(H)$, and by Lemma 3 $L_m(H) \subseteq R(L_m(G))$. Hence $tw \in R(L_m(G))$. This implies that there are $s'$ and $v'$ such that

$$R(s') = t \quad \text{and} \quad R(v') = w \quad \text{and} \quad s't' \in L_m(G).$$

Since $R(s) = R(s') = t$ and by the symmetric structure of the plant under Assumptions (A1), (A2), it can be shown that there exists $v$ such that $R(v) = R(v') = w$ and $sv \in L_m(G)$.

On the other hand,

$$tw \in L_m(\Sigma_{sup})$$

$$\Rightarrow R(s)R(v) \in L_m(\Sigma_{sup})$$

$$\Rightarrow R(sv) \in L_m(\Sigma_{sup})$$

$$\Rightarrow sv \in R^{-1}(L_m(\Sigma_{sup}))$$

$$\Rightarrow sv \in L_m(\Sigma_{sup}).$$

Hence

$$sv \in L_m(\Sigma_{sup}) \cap L_m(G)$$

by which we conclude that $s \in L_m(\Sigma_{sup}) \cap L_m(G)$. □