Distributed Contract Negotiation for Decentralised Supervisory Control beyond Two-Component Architectures

Ana Maria Mainhardt and Anne-Kathrin Schmuck

***Abstract*— We study the control problem of distributed discrete event systems with a privacy aspect. Each component of a system may synchronise with one or more groups of components through different sets of shared events. For each group a component belongs to, its behaviour in terms of its nonshared events (with respect to that group) should remain private. To synthesise decentralised supervisors local to each component, we propose a contract-based negotiation method. A contract describes an agreement among the members of each group. This allows cooperation in disabling shared events, which might not be controllable by all components, in order to guarantee the specifications are met. This work extends our previous results, where only two subsystems with local specifications were considered, to allow more complex architectures and nonlocal specifications. We identify cases where there is no need for coordinators nor the communication of nonshared events, respecting privacy both during the synthesis and the execution of the supervisors, even if for some systems that comes at the cost of maximal permissiveness.**

I. Introduction

Assume-guarantee contracts are a well established paradigm in distributed system design and have a longstanding history – see e.g. [1], [2]. The main idea of contract-based design is to decouple dependent distributed processes by *compatible contracts* which must be locally satisfied by each component to achieve a desired global behaviour. The advantages of contract-based design are threefold: (i) decentralised design, i.e., controllers can be synthesised and executed locally; (ii) information privacy, i.e., apart from what is specified by the contracts, no detailed information about a process' behaviour is shared with the rest of the system; and (iii) decoupled maintenance, as contract-compatible adaptations in a component do not affect others.

Contract-based supervisory synthesis (CBSS) [3], [4] tailors this paradigm to distributed discrete-event systems (DES). Here, processes of a system, also called plant, synchronise via shared events and are controlled by supervisors *local* to each process, or components of a plant. In this context, *compatible contracts* model required behavioural restrictions of all processes solely in terms of events they share. Our previous work [3], [4] introduced a two-component CBSS framework for cosynthesising such compatible contracts and their enforcing local supervisors (schematically depicted in Fig. 1). In this paper, we generalise this approach to multicomponent architectures. This introduces new challenges for privacy preservation, as each plant may synchronise with multiple groups of components through different sets of shared events. Our framework ensures that for each group a component belongs to, its behaviour in terms of its nonshared events (with respect to that group) remains private.

Ensuring nonconflict, i.e., nonblocking global behaviour among the locally supervised processes is a critical step in decentralised DES. In particular for CBSS, where supervisors are designed locally and do not observe private events of other processes they synchronise with over their shared events. In the two-component case, our previous approach [3], [4] identifies and is capable of enforcing sufficient *local conditions* to guarantee *nonconflict by construction* in the context of CBSS (see Fig. 1) – without communicating private events (as e.g. in [5]–[7]) or the utilisation of a *coordinator*, i.e., an additional supervisor that coordinates the control actions of the local ones (see [8]). Unfortunately, in multicomponent systems, circular dependencies often arise, making coordination unavoidable.

Our contribution lies in identifying minimal coordination scopes: when coordination is necessary, we limit it to a small subset of processes, preserving privacy elsewhere. In particular, we introduce basic architectures and combinations thereof that still allow to enforce nonconflict locally, preserving privacy. Moreover, as a byproduct of our ability to handle multicomponent systems, we can incorporate nonlocal specifications and predefined coordinators as additional components without additional formalisation.

Applications of our novel CBSS framework span scenarios where both decentralisation and privacy are essential. For instance, control nodes managing the usage and pricing of private renewable energy sources must keep decision mechanisms confidential, while still requiring coordination to ensure grid stability. Another example involves inter-vehicle coordination in autonomous traffic, where vehicles collaboratively negotiate right-of-way rules based on shared traffic signals, without revealing proprietary driving models or sensor data. Observing page constrains, we limit the exposition in this paper to the theoretical foundations of multicomponent CBSS, leaving experiments to future work.

Related work in distributed supervisory control, e.g. [5]–[7], [9], does not emphasise privacy of nonshared events during synthesis *and* deployment. On the other hand, related work in formal methods typically uses alternating games on graphs to model component interaction, e.g. [10]–[12], where nonconflict is trivially fulfilled. Enforcing nonconflict while preserving privacy is a contribution unique to CBSS.

Both authors are funded through the DFG Emmy Noether grant SCHM 3541/1-1. A. Schmuck is also partially funded through the DFG collaborative research center 389792660 TRR 248 – CPEC.

Both authors are with the Max-Planck Institute for Software Systems, Kaiserslautern, Germany (e-mail: {amainhardt,akschmuck}@mpi-sws.org).

II. Preliminaries

A. Languages and Automata Basics

Strings. Let Σ be a finite *alphabet*, which is a nonempty set of symbols $\sigma \in \Sigma$ representing the events of a DES. The *Kleene-closure* Σ^* is the set of finite strings $s = \sigma_1\sigma_2\cdots\sigma_n$, with $n \in \mathbb{N}$ and $\sigma_i \in \Sigma$, including the *empty string* $\epsilon \in \Sigma^*$, with $\epsilon \notin \Sigma$. If, for two strings $s, r \in \Sigma^*$, there exists $t \in \Sigma^*$ such that $s = rt$, we say r is a *prefix* of s, and write $r \leq s$.
Sets. For any two sets A and B, denote by $A \times B$ and $A \backslash B$, respectively, their Cartesian product and set difference. The cardinality of A is denoted by $|A|$. If $A = \{a_1, \ldots, a_n\}$, we write $A = \{a_i\}_I$ for $I = \{1, \ldots, n\}$. If a_i is any constant value a for all i, we write $A = \{a\}_I$.
Languages. A *language* over Σ is a subset $L \subseteq \Sigma^*$. The *prefix* of a language $L \subseteq \Sigma^*$ is defined by $\overline{L} := \{r \in \Sigma^* \mid \exists s \in L : r \leq s\}$. The prefix operator is also referred to as the *prefix-closure*, and a language L is *closed* if $L = \overline{L}$. A language K is *relatively closed with respect to* L, or simply *L-closed*, if $K = \overline{K} \cap L$. The prefix operator distributes over arbitrary unions of languages. However, for the intersection of two languages L and M, we have $\overline{L \cap M} \subseteq \overline{L} \cap \overline{M}$. If equality holds, L and M are said to be *nonconflicting*.
Projections. Given alphabets Σ and $\Sigma_i \subseteq \Sigma$, the *natural projection* $P_i : \Sigma^* \to \Sigma_i^*$ is defined recursively by: (i) $P_i(\epsilon) = \epsilon$, and (ii) for all $\sigma \in \Sigma$ and $s \in \Sigma^*$, if $\sigma \in \Sigma_i$, then $P_i(s\sigma) = P_i(s)\sigma$, otherwise $P_i(s\sigma) = P_i(s)$. We define the *inverse projection* $P_i^{-1} : \Sigma_i^* \to 2^{\Sigma^*}$ by $P_i^{-1}(s_i) = \{s \in \Sigma^* \mid P_i(s) = s_i\}$ for all $s_i \in \Sigma_i^*$. These functions can be extended from strings to languages, with $P_i : 2^{\Sigma^*} \to 2^{\Sigma_i^*}$ defined such that, for any $L \subseteq \Sigma^*$, $P_i(L) = \{s_i \in \Sigma_i^* \mid \exists s \in L : P_i(s) = s_i\}$. In turn, $P_i^{-1} : 2^{\Sigma_i^*} \to 2^{\Sigma^*}$ is given by $P_i^{-1}(L_i) = \{s \in \Sigma^* \mid P_i(s) \in L_i\}$ for any $L_i \subseteq \Sigma_i^*$.
Automata. A *deterministic finite automaton* (*automaton* for short) is a tuple $\mathbf{\Lambda} = (Q, \Sigma, \delta, q_0, Q_\mathrm{m})$, with finite *state set* Q, *initial state* $q_0 \in Q$, *marked states* $Q_\mathrm{m} \subseteq Q$, and *deterministic transition function* $\delta : Q \times \Sigma \to Q$, which can be partial and also be viewed as a relation $\delta \subseteq Q \times \Sigma \times Q$. We identify δ with its extension to the domain $Q \times \Sigma^*$, or as $\delta \subseteq Q \times \Sigma^* \times Q$. Let $\delta(q, s)!$ indicate that δ is defined for $q \in Q$ and $s \in \Sigma^*$; for all $q \in Q$, we have $\delta(q, \epsilon) = q$; for $s \in \Sigma^*$ and $\sigma \in \Sigma$, we have $\delta(q, s\sigma)!$, with $\delta(q, s\sigma) = \delta(\delta(q, s), \sigma))$, if and only if $\delta(q, s)!$ and $\delta(\delta(q, s), \sigma))!$. We say $\mathbf{\Lambda}$ is nonempty if $Q \neq \emptyset$.
Reachability and Nonblockingness. A state $q \in Q$ is *reachable* if there exists $s \in \Sigma^*$ such that $q = \delta(q_0, s)$, and it is *coreachable* if there exists $s \in \Sigma^*$ such that $\delta(q, s) \in Q_\mathrm{m}$. Non coreachable states are also referred to as *blocking states*. If all reachable states in $\mathbf{\Lambda}$ are coreachable, then $\mathbf{\Lambda}$ is *nonblocking*. Moreover, $\mathbf{\Lambda}$ is called *reachable* (respectively *coreachable*) if all states are reachable (resp. coreachable), and $\mathbf{\Lambda}$ is called *trim* if it is reachable and coreachable.
Semantics. For $\mathbf{\Lambda} = (Q, \Sigma, \delta, q_0, Q_\mathrm{m})$, we associate the *generated language* $\mathcal{L}(\mathbf{\Lambda}) := \{s \in \Sigma^* \mid \delta(q_0, s)!\}$ and the *marked language* $\mathcal{L}_m(\mathbf{\Lambda}) := \{s \in \Sigma^* \mid \delta(q_0, s) \in Q_\mathrm{m}\}$. These two languages are the *semantics* of $\mathbf{\Lambda}$, and $\mathbf{\Lambda}$ *recognizes*, or *accepts*, the language $\mathcal{L}_m(\mathbf{\Lambda})$. Moreover, we say two automata $\mathbf{\Lambda}_1$ and $\mathbf{\Lambda}_2$ are *equivalent* if their semantics coincide, i.e., we write $\mathbf{\Lambda}_1 \equiv \mathbf{\Lambda}_2$ iff $\mathcal{L}(\mathbf{\Lambda}_1) = \mathcal{L}(\mathbf{\Lambda}_2)$ and $\mathcal{L}_m(\mathbf{\Lambda}_1) = \mathcal{L}_m(\mathbf{\Lambda}_2)$; if they are not equivalent, we write $\mathbf{\Lambda}_1 \not\equiv \mathbf{\Lambda}_2$. Note that $\mathbf{\Lambda}$ is nonblocking if and only if $\mathcal{L}(\mathbf{\Lambda}) = \overline{\mathcal{L}_m(\mathbf{\Lambda})}$. To simplify notation, we use boldface characters, e.g. $\mathbf{\Lambda}$, to denote an automaton, and the corresponding normal character, e.g. Λ, for its marked language.
Product and Composition. The *synchronous product* of languages $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, where Σ_i, for $i \in \{1, 2\}$, are arbitrary alphabets, is defined as $L_1 \parallel L_2 := P_1^{-1}(L_1) \cap P_2^{-1}(L_2)$ over $\Sigma_1 \cup \Sigma_2$. Given two automata $\mathbf{\Lambda}_i$ over Σ_i, their *synchronous composition* $\mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2$ is defined such that $\mathcal{L}(\mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2) = \mathcal{L}(\mathbf{\Lambda}_1) \parallel \mathcal{L}(\mathbf{\Lambda}_2)$ and $\mathcal{L}_m(\mathbf{\Lambda}_1 \parallel \mathbf{\Lambda}_2) = \Lambda_1 \parallel \Lambda_2$. That is, only shared events $\sigma \in \Sigma^\mathrm{s} = \Sigma_1 \cap \Sigma_2$ require synchronisation of the corresponding transitions.
Observer Automaton. For an automaton $\mathbf{\Lambda}$ over $\Sigma = \Sigma^\mathrm{s} \dot\cup \Sigma^\mathrm{p}$, its observer $\mathbf{O}_{\mathbf{\Lambda}, \Sigma^\mathrm{s}}$ is an automaton over Σ^s built in such a way that it generates and accepts the languages $P_\mathrm{s}(\mathcal{L}(\mathbf{\Lambda}))$ and $P_\mathrm{s}(\Lambda)$, respectively, where $P_\mathrm{s} : \Sigma^* \to \Sigma^{\mathrm{s}*}$. Its construction and properties are described in [13].

B. Supervisory Control

For a more didactic coverage, see the textbooks [13] and [8].
Plant Model. A plant is a system to be supervised, modelled as an automaton $\mathbf{M}$ over Σ. Typically $\Sigma := \Sigma_\mathrm{c} \dot\cup \Sigma_\mathrm{uc}$, where Σ_c is the set of *controllable* events – those that can be prevented from happening (i.e. disabled) – and Σ_uc is the set of *uncontrollable* ones – that cannot be disabled.
Plantified specification. For any prefix-closed specification language E over $\Sigma' \subseteq \Sigma$, the language $K = M \parallel E$ describes the *desired behaviour* of $\mathbf{M}$, and it is M-closed. We call a *plantified specification* of $\mathbf{M}$ an automaton $\mathbf{K}$ that accepts K and such that: (i) $\mathcal{L}(\mathbf{K}) \subseteq \mathcal{L}(\mathbf{M})$, and (ii) for all $s \in \overline{K}$, and for all $\sigma \in \Sigma$, $s\sigma \in \mathcal{L}(\mathbf{K})$ if $s\sigma \in \mathcal{L}(\mathbf{M})$. We note that $\mathbf{K}$ is generally blocking. Further, *specification* and *plantified specification* are not interchangeable concepts.
Controllability. A language L over Σ is *controllable* with respect to $\mathcal{L}(\mathbf{M})$ and Σ_uc if, for all $\sigma \in \Sigma_\mathrm{uc}$, $s \in \overline{L} \wedge s\sigma \in \mathcal{L}(\mathbf{M}) \Rightarrow s\sigma \in \overline{L}$. Define the set $\mathcal{C}(L) := \{L' \subseteq L \mid L' \text{ is controllable w.r.t. } \mathcal{L}(\mathbf{M}) \text{ and } \Sigma_\mathrm{uc}\}$, whether L is controllable or not. This set is nonempty, since the empty language is trivially controllable. As controllability is closed under union of languages, it can be shown that the supremum element of $\mathcal{C}(L)$, denoted $\sup\mathcal{C}(L)$, is given by $\bigcup_{L' \in \mathcal{C}(L)} L'$ and is controllable, i.e., belongs to $\mathcal{C}(L)$.
Supervisor. A *supervisor* for a plant $\mathbf{M}$ with alphabet $\Sigma := \Sigma_\mathrm{c} \dot\cup \Sigma_\mathrm{uc}$ is a mapping $f : \mathcal{L}(\mathbf{M}) \to \Gamma$, where $\Gamma := \{\gamma \subseteq \Sigma \mid \Sigma_\mathrm{uc} \subseteq \gamma\} \subseteq 2^\Sigma$ is the set of *control patterns*. Let us denote by $f/\mathbf{M}$ the plant $\mathbf{M}$ under supervision of f. The generated language of $f/\mathbf{M}$ is defined recursively such that $\epsilon \in \mathcal{L}(f/\mathbf{M})$ and, for all $s \in \Sigma^*$ and $\sigma \in \Sigma$, $s\sigma \in \mathcal{L}(f/\mathbf{M})$ iff (i) $s \in \mathcal{L}(f/\mathbf{M})$, (ii) $s\sigma \in \mathcal{L}(\mathbf{M})$, and (iii) $\sigma \in f(s)$. This induces the marked language $\mathcal{L}_m(f/\mathbf{M}) := \mathcal{L}(f/\mathbf{M}) \cap M$, which is controllable by definition. Note that $\mathcal{L}(f/\mathbf{M})$ is closed and $\overline{\mathcal{L}(f/\mathbf{M})} \subseteq \mathcal{L}(\mathbf{M})$. We call f nonblocking if $\mathcal{L}(f/\mathbf{M}) = \overline{\mathcal{L}_m(f/\mathbf{M})}$. In this case, the closed loop can be

represented by a trim automaton $\mathbf{S}$ that accepts $\mathcal{L}_m(f/\mathbf{M})$; we then say that $\mathbf{S}$ *realises* $f/\mathbf{M}$ and denote this by $\mathbf{S} \sim f$.

Supervisor Synthesis Problem. Given a plant $\mathbf{M}$ and a plantified specification $\mathbf{K}$ over $\Sigma := \Sigma_c \dot\cup \Sigma_{uc}$, the control problem is to synthesise the *maximally permissive* supervisor f that respects the specification – i.e., $\mathcal{L}_m(f/\mathbf{M}) = \sup\mathcal{C}(K)$, with controllability taken with respect to $\mathcal{L}(\mathbf{M})$ and Σ_{uc} – and that imposes a nonblocking closed-loop behaviour – i.e., $\mathcal{L}(f/\mathbf{M}) = \overline{\mathcal{L}_m(f/\mathbf{M})}$. To synthesise f, it is possible to compute a trim automaton $\mathbf{S} \sim f$ by manipulating the plantified specification $\mathbf{K}$ (see [13, p.186]).

C. Cooperative Decentralised Supervisory Control

Distributed Plant. A plant is a finite set $\mathcal{M} = \{\mathbf{M}_i\}_I$, for $I = \{1, \ldots, n\}$. Each *local component* $i \in I$ is modelled as an automaton $\mathbf{M}_i$ and equipped with a plantified specification $\mathbf{K}_i$, both over the *local* alphabet Σ_i. The *global* alphabet of $\mathcal{M}$ is given by $\Sigma = \bigcup_i \Sigma_i$. We use the convention that i, j are always taken from I. For any $i \neq j$, denote $\Sigma_i \cap \Sigma_j$ by $\Sigma_{i,j}$. The alphabet of *shared* events of i is then $\Sigma_i^{s} = \bigcup_{j \neq i} \Sigma_{i,j}$. A local alphabet is partitioned into $\Sigma_i = \Sigma_i^{p} \dot\cup \Sigma_i^{s}$, where Σ_i^{p} is the set of *private* events of i. We assume that Σ_i^{p} may contain controllable and uncontrollable events, meaning $\Sigma_i^{p} = \Sigma_{i,c}^{p} \dot\cup \Sigma_{i,uc}^{p}$. We denote by P_{is} and P_i the natural projections $P_{is} : \Sigma_i^* \to \Sigma_i^{s*}$ and $P_i : \Sigma^* \to \Sigma_i^*$.

Cooperative Supervisor Synthesis. Following [3]–[5], components are *not* required to agree on the controllability status of the events they share. However, for simplicity[1], we assume that for any component i, and for any $\sigma \in \Sigma_i^{s}$, there is j (possibly $j = i$) such that σ is controllable by j. Thus, we have that $\Sigma_i^{s} = \Sigma_{i,c}^{s} \dot\cup \Sigma_{i,cp}^{s}$, where $\Sigma_{i,c}^{s}$ are shared events controllable by i, and $\Sigma_{i,cp}^{s}$ are shared events uncontrollable by i, but controllable by some $j \neq i$. The subscript cp stands for *cooperation*. Similar to the classical supervisor synthesis problem defined in Sec. II-B, in CBSS we synthesise local supervisors f_i for the components $\mathbf{M}_i$ by computing trim automata $\mathbf{S}_i$ obtained from $\mathbf{K}_i$. Moreover, to enhance permissiveness, these supervisors should assist each other in achieving joint control, as events in $\Sigma_{i,cp}^{s}$ are globally controllable (see [4]). In order to achieve the desired cooperation, we then utilise the following conditions for controllability and a local synthesis procedure CSYNTH based on the uncontrollable event set $\Sigma_{i,uc}^{p}$, instead of the actual locally uncontrollable event set $\Sigma_{i,uc} = \Sigma_{i,uc}^{p} \dot\cup \Sigma_{i,cp}^{s}$.

Condition 1: For any automaton $\mathbf{\Lambda}_i$ over Σ_i, we define the following requirements:

1) $\mathcal{L}(\mathbf{\Lambda}_i) \subseteq \mathcal{L}(\mathbf{M}_i)$ and $\Lambda_i \subseteq M_i$;
2) $(\forall s \in \overline{\Lambda_i},\ \forall \sigma \in \Sigma_{i,uc}^{p})\ \big(s\sigma \notin \overline{\Lambda_i}$ and $s\sigma \in \mathcal{L}(\mathbf{M}_i)\big) \to s\sigma \in \mathcal{L}(\mathbf{\Lambda}_i)$.

Definition 1: For an automaton $\mathbf{S}_i^0$ over $\Sigma_{i,c}^{p} \dot\cup \Sigma_{i,uc}^{p} \dot\cup \Sigma_{i,c}^{s} \dot\cup \Sigma_{i,cp}^{s}$, the *cooperative* synthesis function CSYNTH is defined such that $\text{CSYNTH}(\mathbf{S}_i^0)$ is a trim automaton accepting the language $\sup\mathcal{C}(S_i^0)$ with respect to $\mathbf{S}_i^0$ and $\Sigma_{i,uc}^{p}$. □

[1]See [4] on how two-component plants can be treated when this assumption does not hold. While the approach from [4] can be directly incorporated in the multicomponent setting discussed in this paper, we refrain from doing so to simplify the approach as we observe page constraints.

Remark 1: Let $\mathbf{S}_i^0$ be an automaton that satisfies Cond. 1 (e.g., the plantified specification $\mathbf{K}_i$). Then $\sup\mathcal{C}(S_i^0)$ with respect to $\mathbf{M}_i$ and $\Sigma_{i,uc}^{p}$ is the same as with respect to $\mathbf{S}_i^0$ and $\Sigma_{i,uc}^{p}$. Moreover, $\text{CSYNTH}(\mathbf{S}_i^0)$ also satisfies Cond. 1, and CSYNTH can be implemented using existing techniques.

Local Property for Global Nonblockingness. Let $\mathbf{S}_i$ be automata used to extract local supervisors f_i. As in the case of CBSS for two components ([3], [4]), here we also require S_i to satisfy the following local property so that f_i impose a nonblocking global behaviour.

Definition 2 ([4]): The language S_i is *unambiguous* with respect to $\Sigma^g \subseteq \Sigma_i$ if, for any $s, s' \in \overline{S_i}$ such that $P_{ig}(s) = P_{ig}(s')$, where $P_{ig} : \Sigma_i^* \to \Sigma^{g*}$, we have that:

1) $(\forall \sigma \in \Sigma^g)\ s\sigma \in \overline{S_i} \ \Rightarrow (\exists s'' \in \Sigma_i^*)$
   $s' \leq s''$, $P_{ig}(s'') = P_{ig}(s')$ and $s''\sigma \in \overline{S_i}$; and
2) $s \in S_i \Rightarrow (\exists s'' \in \Sigma_i^*)$
   $s'' \leq s'$ or $s' < s''$, $P_{ig}(s'') = P_{ig}(s')$ and $s'' \in S_i$; □

In [4] we introduce a procedure called ENFORCERU which locally checks if unambiguity is satisfied, and if not, enforces a slightly stronger version called *relative unambiguity*, as unambiguity is not closed under union of languages.

Remark 2: For an automaton $\mathbf{S}_i^0$ satisfying Cond. 1, the automaton $\text{ENFORCERU}(\mathbf{S}_i^0, \Sigma^g)$ also satisfies Cond. 1, and its accepted language is unambiguous with respect to Σ^g. Moreover, if $\mathbf{S}_i^0$ accepted language is already unambiguous with respect to Σ^g, then $\mathbf{S}_i^0 \equiv \text{ENFORCERU}(\mathbf{S}_i^0, \Sigma^g)$. □

Signalling Bits. In order to extract from $\mathbf{S}_i$ local supervisors f_i that actually exhibit the intended cooperation whenever required – i.e. when $\Sigma_{i,cp}^{s} \neq \emptyset$ – as in [4] we propose the use of a signalling bit for each shared event $\sigma \in \bigcup_{i \in I} \Sigma_{i,cp}^{s}$. We define $\Sigma^{d} = \{\mathrm{d}^\sigma \,|\, \sigma \in \Sigma_{i,cp}^{s} \text{ and } i \in I\}$, where d^σ models the bit corresponding to σ having value one, which represents a request for supervisor i that controls σ to disable it. We further define $\Sigma_{i\to}^{d} = \{\mathrm{d}^\sigma \,|\, \sigma \in \Sigma_{i,cp}^{s}\}$ and $\Sigma_{\to i}^{d} = \Sigma^{d} \cap \{\mathrm{d}^\sigma \,|\, \sigma \in \Sigma_{i,c}^{s}\}$ as the requests that can be made by or for i, corresponding to the bits that can be written or read by i, respectively. The use of these bits does respect the given privacy concerns, as discussed in more details in [4]. It is different from what is called *communication* in the literature – e.g. in [6] – where a subset of the private events needs to be always observed by other supervisors, i.e., every occurrence of such events is communicated. Here, when supervisor i reads one in the bit relative to σ, it cannot know who set that bit to one, nor which (or how many) of their private events triggered this. Moreover, in principle i cannot even infer this information, as our approach does not share – apart from the contract that is only described in terms of the shared events – the local model of each subsystem, neither during the synthesis nor the execution of the supervisors.

Global vs. Local Supervisor Synthesis Problem. Summarising the above discussion, this paper tackles the following decentralised supervisor synthesis problem:

Problem 1: Find *local* supervisors $f_i : \mathcal{L}(\mathbf{M}_i) \times 2^{\Sigma_{\to i}^{d}} \to \Gamma_i$ such that the *global* closed-loop behaviour satisfies the specifications and is *nonblocking*, that is, $\|_i\ \mathcal{L}_m(f_i/\mathbf{M}_i) \subseteq \|_i\ K_i$ and $\|_i\ \mathcal{L}(f_i/\mathbf{M}_i) = \overline{\|_i\ \mathcal{L}_m(f_i/\mathbf{M}_i)}$, where the languages $\mathcal{L}_{(m)}(f_i/\mathbf{M}_i)$ are defined from f_i as in Sec. II-B.

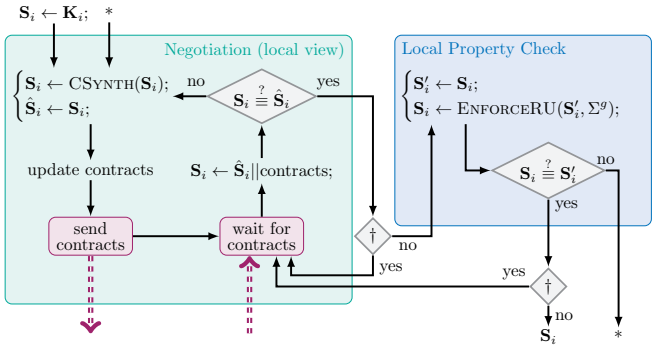

Fig. 1. Simplified *local overview* of CBSS for component i. Functions CSynth and EnforceRU are as in Def. 1 and Rem. 2, respectively. Symbol † stands for "Have other subsystems updated their contracts?". Synthesis is initialised with plantified specifications $\mathbf{K}_i$, and terminates when all components locally converged – i.e., when they are locally nonblocking, satisfy the local property, and have compatible contracts. Local supervisors f_i are extracted from $\mathbf{S}_i$ upon termination (see Sec. VIII-A).

III. Outline and Contribution

This paper extends existing contract-based supervisor synthesis (CBSS) frameworks [3], [4], as depicted in Fig. 1, from two to multicomponent architectures. What distinguishes CBSS from other solutions to Problem 1 is its strong emphasis on privacy – both during synthesis and during execution, components have solely access to their observed interaction with other components via *shared events*. In order to still remain permissive, CBSS iteratively refines *contracts* which represent cooperative *agreements*.

Contract Negotiation. In multiprocesses architectures, components might share different set of events, let us say Σ^g, with different groups g of components (formally defined in Sec. IV). Thus, in contrast to [3], [4], each component i needs to negotiate different contracts $\mathbf{C}_i^g$, one for each group g that i belongs to. As the behaviour of a process in terms of its private events (with respect to each group) should remain private, we define $\mathbf{C}_i^g := \mathbf{O}_{\mathbf{S}_i,\Sigma^g}$[2]. For two components i, j with shared events $\Sigma_{i,j} = \Sigma^g$, we say they have *compatible contracts* iff $\mathbf{C}_i^g \equiv \mathbf{C}_j^g$. For the synthesis of the local supervisors, we then iteratively refine the automata $\mathbf{S}_i$, update their contracts $\mathbf{C}_i^g$, and negotiate these contracts among members of each group, as illustrated by the *Negotiation* box in Fig. 1. These iterations terminate when contract-compatibility and local nonblockingness of $\mathbf{S}_i$ are achieved, as formalised in Sec. V. In particular, we show in Sec. VI that negotiation is permissive and does not overly restrict the behaviour of the plant.

Ensuring Nonconflict. After negotiation, we still need to check whether each language S_i locally satisfies the property of unambiguity, as introduced in Sec. II-C. If not satisfied, then it is enforced over $\mathbf{S}_i$, which might compromise local nonblockingness, controllability and contract-compatibility, requiring further negotiation, as illustrated by the *Local Property Check* box in Fig. 1. Moreover, enforcing this property might imply sacrificing maximally permissiveness in favour of privacy. In Sec. VII we identify multicomponent architectures for which unambiguity, in the context of CBSS, is sufficient to guarantee nonconflicting supervisors.

Enhanced Flexibility. By considering beyond-two-component architectures, we are able to deal with nondecomposable specifications spanning multiple components, as we explain in Sec. IV-A. This scenario was not tackled in [3], [4], where only local specifications were allowed. Moreover, in Sec. VII-C we show how coordinators can be obtained under our CBSS approach in systems with architectures that require trading privacy for conflict resolution.

Completing the Picture. As a final result of all contributions discussed above, we show in Sec. VIII that cooperative local supervisors extracted from the output of our novel multicomponent CBSS framework indeed solve Problem 1 with a strong emphasis on permissiveness and privacy.

IV. Component Architectures

This section formalises how component architectures induce groups for negotiation, and explains the resulting flexibility of our framework by showing how nondecomposable (nonlocal) specifications can be incorporated as additional components in the CBSS framework.

There are multiple distinct ways how components can be combined into groups based on the events they share, and such that the final result of CBSS is provably correct. In this paper, we choose a group definition more convenient for the proofs. Concretely, we form groups such that every possible pair of elements in a group shares exactly the same set of events, and all components in a system are associated with as many groups as possible, as long as it respects the previous rule. In Fig. 2, 3 and 4, we illustrate few possible architectures of plants based on the groups they can form.

Definition 3: For a plant $\mathcal{M} = \{\mathbf{M}_i\}_I$, and for all $g = (I^g, \Sigma^g) \in 2^I \times (2^\Sigma \backslash \{\emptyset\})$, we say g is a *group* if and only if: (i) for all $i, j \in I^g$ we have $\Sigma_{i,j} = \Sigma^g$, and (ii) for all $i \in I \backslash I^g$, there is $j \in I^g$ such that $\Sigma_{i,j} \neq \Sigma^g$. Define the projections $P_{ig} : \Sigma_i^* \to \Sigma^{g*}$ for all $g \in I^g$. Denote by $\mathcal{G} = \{g_k\}_K$, with $K = \{1, \ldots, m\}$, the set of all groups for the plant $\mathcal{M}$. Moreover, denote by $\mathcal{G}_i = \{g \in \mathcal{G} \mid i \in I^g\}$ the set of all groups to which i belongs, for all $i \in I$.

A. Plants with Nonlocal Specifications

Within the framework presented here, we can also impose nonlocal specifications, which cannot be decomposable into local ones. This is a consequence of extending our results from [4] to systems formed by more than two components. Consider then any prefix-closed specification language E over $\Sigma' \subseteq \Sigma$ that cannot be decomposable into local alphabets Σ_i. Denote by $\mathbf{E}$ the trim automaton that accepts E. Then, in order to obtain a supervisor $\mathbf{S}_E$ that guarantees E is satisfied, we need to add a component $\mathbf{M}_E$ in the plant

[2]In our prior work [3], [4] we define the contracts as $\mathbf{C}_i := \mathbf{O}_{\mathbf{S}_j,\Sigma^s}$. Here, in contrast, we define $\mathbf{C}_i^g := \mathbf{O}_{\mathbf{S}_i,\Sigma^g}$, such that the contract with subscript i is not the observer of other component j, but of i itself. The reason is the following. In [3], [4] we consider only two processes, so there is only one other component $j \neq i$, and only one group with shared alphabet Σ^s instead of Σ^g. Here, each component exchanges contracts with not only one, but possibly multiple other components from groups g, which are represented by the superscript g in the contract.

$\mathcal{M}$, as follows. Define $\mathbf{M}_E$ as the single state automaton that accepts and generates Σ'^*, and use the product $\mathbf{M}_E \parallel \mathbf{E}$ as its plantified specification $\mathbf{K}_E$. Note that defining $\mathbf{M}_E$ this way guarantees controllability is respected.

V. Multicomponent CBSS

This section presents our novel multicomponent contract-based supervisor synthesis (CBSS) framework, as schematically depicted in Fig. 1 and formalised in Proc. 1 and Proc. 2. The CBSS procedure starts by calling NEGOTIATION. In the first round of NEGOTIATION, the automata $\mathbf{S}_i^0$ of all components i are refined by the local synthesis procedure CSYNTH. Next, contracts $\mathbf{C}_i^g$ are drawn for all components i, and all groups g to which i belongs. All these contracts are then exchanged among members of the same group, for all groups $g \in \mathcal{G}$. From the second round on, local synthesis CSYNTH is redone and new contracts $\mathbf{C}_i^g$ are drawn again only for the automata $\mathbf{S}_i$ on which the assimilation of the received contracts altered the previous computation of CSYNTH. Then, only the components that are in the same group as a component that produced a new contract will be altered again. Convergency occurs when the exchange of contracts do not affect the outcome of CSYNTH for all components i. NEGOTIATION then outputs all the automata $\mathbf{S}_i$, and also a variable $changed_i$ indicating whether the output $\mathbf{S}_i$ is equivalent to the input $\mathbf{S}_i^0$.

In CBSS, the set *WhereToEnforceRU* indicates all possible pairs of components and groups they belongs to, as the local property of unambiguity should be checked and, if needed, enforced for all these pairs. After NEGOTIATION, CBSS picks and removes a pair (i, g) from the set *WhereToEnforceRU* with the function *Popp*. Then, for this pair, ENFORCERU checks if S_i satisfies unambiguity with respect to Σ^g. If it does, *enforcedOnOne* is set to false (as the property did not need to be enforced), and a new pair is chosen, until either no pair is left – meaning the property is satisfied locally by all components, and with respect to all groups they belongs to – or until the property is enforced for some pair. In the first case, CBSS terminates; in the latter case, NEGOTIATION is called again.

Procedure 1 CBSS

Require: Automata $\{\mathbf{S}_i^0\}_I$.
1: *WhereToEnforceRU* $\leftarrow \{(i,g) \in I \times \mathcal{G} | i \in g\}$
2: $\{S_i, changed_i, synth_i\}_I \leftarrow \{S_i^0, \texttt{false}, \texttt{true}\}_I$
3: **repeat**
4: $\quad \{S_i, changed_i\}_I \leftarrow$ NEGOTIATION$(\{S_i, synth_i\}_I)$
5: $\quad \{synth_i\}_I \leftarrow \{\texttt{false}\}_I$
6: $\quad$ **if** $\exists i$ such that $\mathbf{S}_i$ is the empty automaton **then return** $\{\mathbf{S}_i\}_I$
7: $\quad$ **for** $i \in I$ such that $changed_i$ and $g \in \mathcal{G}_i$ **do**
8: $\quad\quad$ *WhereToEnforceRU* $\leftarrow$ *WhereToEnforceRU* $\cup \{(i,g)\}$
9: $\quad$ *enforcedOnOne* $\leftarrow \texttt{false}$, $i \leftarrow -1$
10: $\quad$ **while** *WhereToEnforceRU* $\neq \emptyset$ and $\neg$*enforcedOnOne* **do**
11: $\quad\quad$ $(i,g) \leftarrow$ *Popp*(*WhereToEnforceRU*)
12: $\quad\quad$ $(\mathbf{S}_i, enforcedOnOne) \leftarrow$ ENFORCERU$(\mathbf{S}_i, \Sigma^g)$
13: $\quad\quad$ **if** *enforcedOnOne* **then** $synth_i \leftarrow \texttt{true}$
14: $\quad$ **if** $\exists i$ such that $\mathbf{S}_i$ is the empty automaton **then return** $\{\mathbf{S}_i\}_I$
15: $\quad$ **if** $\neg$*enforcedOnOne* **then** $i \leftarrow -1$
16: **until** $i \neq -1$
17: **return** $\{\mathbf{S}_i\}_I$

Procedure 2 NEGOTIATION

Require: Automata $\{\mathbf{S}_i^0\}_I$. **Optional:** Boolean variables $\{synth_i\}_I$ (otherwise, assume $\{synth_i\}_I = \{\texttt{true}\}_I$).
1: $\{\mathbf{S}_i, \hat{\mathbf{S}}_i\}_I \leftarrow \{\mathbf{S}_i^0, \emptyset\}_I$, $\{setOfNewC^g\}_\mathcal{G} \leftarrow \{\emptyset\}_\mathcal{G}$
2: $\{compare_i, drawC_i, changed_i\}_I \leftarrow \{\texttt{false}, \texttt{false}, \texttt{false}\}_I$
3: $cond \leftarrow \texttt{True}$
4: **while** $cond$ **do**
5: $\quad cond \leftarrow \texttt{False}$
6: $\quad$ **for** $i \in I$ **do**
7: $\quad\quad$ **if** $compare_i$ **then**
8: $\quad\quad\quad synth_i \leftarrow (\hat{\mathbf{S}}_i \not\equiv \mathbf{S}_i)$, $compare_i \leftarrow \texttt{False}$
9: $\quad\quad\quad changed_i \leftarrow changed_i$ or $synth_i$
10: $\quad\quad$ **if** $synth_i$ **then**
11: $\quad\quad\quad \mathbf{S}_i' \leftarrow$ CSYNTH$(\mathbf{S}_i)$, $cond \leftarrow \texttt{True}$
12: $\quad\quad\quad$ **if** $\mathbf{S}_i' \neq \mathbf{S}_i$ **then**
13: $\quad\quad\quad\quad \mathbf{S}_i \leftarrow \mathbf{S}_i'$, $changed_i \leftarrow \texttt{True}$
14: $\quad\quad\quad\quad$ **if** $\mathbf{S}_i$ is the empty automaton **then**
15: $\quad\quad\quad\quad\quad$ **return** $(\{\mathbf{S}_i, changed_i\}_I)$
16: $\quad\quad\quad$ **for** $g \in \mathcal{G}_i$ **do**
17: $\quad\quad\quad\quad \mathbf{C}_i^g \leftarrow \mathbf{O}_{\mathbf{S}_i, \Sigma^g}$, $setOfNewC^g \leftarrow setOfNewC^g \cup \{\mathbf{C}_i^g\}$
18: $\quad\quad\quad synth_i \leftarrow \texttt{False}$,
19: $\quad\quad \hat{\mathbf{S}}_i \leftarrow \mathbf{S}_i$
20: $\quad$ **for** $g \in \mathcal{G}$ such that $setOfNewC^g \neq \emptyset$ **do**
21: $\quad\quad newC^g \leftarrow \|_{\mathbf{C} \in setOfNewC^g} \mathbf{C}$
22: $\quad\quad$ **for** $i \in I^g$ **do**
23: $\quad\quad\quad \mathbf{S}_i \leftarrow \mathbf{S}_i \parallel newC^g$, $compare_i \leftarrow \texttt{True}$
24: $\quad\quad\quad$ **if** $\mathbf{S}_i$ is the empty automaton **then**
25: $\quad\quad\quad\quad$ **return** $(\{\mathbf{S}_i, changed_i\}_I)$
26: $\quad\quad setOfNewC^g \leftarrow \emptyset$
27: **return** $(\{\mathbf{S}_i, changed_i\}_I)$

VI. Maximal Permissiveness of Negotiation

This section shows that the desired maximal permissiveness of contract negotiation carries over from the two- to the multicomponent setting.

Lemma 1: Let $\{\mathbf{S}_i^0\}_I$ be automata satisfying Cond. 1, and $\{\mathbf{S}_i\}_I$ be the outputs of Proc. 2 with $\{\mathbf{S}_i^0\}_I$ as its inputs, namely, $(\{\mathbf{S}_i\}_I) =$ NEGOTIATION$(\{\mathbf{S}_i^0\}_I)$. Then each automaton in $\{\mathbf{S}_i\}_I$ is trim and also satisfies Cond. 1. Moreover, for all $g \in \mathcal{G}$, there is a contract $\mathbf{C}^g$ over Σ^g such that $\mathbf{C}^g \equiv \mathbf{C}_i^g = \mathbf{O}_{\mathbf{S}_i, \Sigma^g}$ for all $i \in I^g$. □

Proof: It is easy to show by induction that the automata $\{\mathbf{S}_i\}_I$ satisfy Cond. 1, as the only operations performed on $\mathbf{S}_i$ during NEGOTIATION are CSYNTH and the product by contracts. CSYNTH, by Sup. Rem. 1, preserves this condition, and the product by contracts $\mathbf{C}^g$ preserves it too, as the synchronisation only happens over shared and not private events Σ_i^p. Denote by $\hat{\mathbf{S}}_i^k$ and $\mathbf{S}_i^k$ the values of $\hat{\mathbf{S}}_i$ and $\mathbf{S}_i$ at the end of the k-th round of negotiation, i.e., at line 19 in NEGOTIATION, for $k \geq 1$. Given the initial value of the boolean variables in the procedure, all components are forced to exchange contracts among the members of the groups they belong to. Thus, it is easy to show by induction on the number of rounds k that, for all $i \in I$, for all $g \in \mathcal{G}_i$, and for all $j \in I^g$, $\mathbf{S}_i^k = \hat{\mathbf{S}}_i^k \parallel \big(\|_{g \in \mathcal{G}_i} \|_{j \in g} \mathbf{O}_{\hat{\mathbf{S}}_j^k, \Sigma^g} \big)$. To see that, just note that in a round k, either $\mathbf{S}_i^k$ receives a new contract $\mathbf{O}_{\hat{\mathbf{S}}_j^k, \Sigma^g}$, if $\hat{\mathbf{S}}_j^k \neq \hat{\mathbf{S}}_j^{k-1}$, or it does not, because $\hat{\mathbf{S}}_j^k$ did not change from the previous round, i.e., $\hat{\mathbf{S}}_j^k = \hat{\mathbf{S}}_j^{k-1}$, so $\mathbf{S}_i^k$ already incorporates that contract. Finally, as the fix point is reached when $\hat{\mathbf{S}}_i^k = \mathbf{S}_i^k$ holds, for some $k > 1$, and for all $i \in I$, we have that $P_{jg}\mathcal{L}(\mathbf{S}_j) \subseteq P_{ig}\mathcal{L}(\mathbf{S}_i)$ and

$P_{jg}S_j \subseteq P_{ig}S_i$. As this is true for all $i \in I$, we have that $P_{jg}\mathcal{L}(\mathbf{S}_j) = P_{ig}\mathcal{L}(\mathbf{S}_i)$ and $P_{jg}S_j = P_{ig}S_i$, for all $g \in \mathcal{G}_i$, and for all $j \in I^g$. ■

With this, we can prove the main result of this section, namely that NEGOTIATION does not remove any more states and transitions than the necessary to obtain the supremal controllable sublangage of the composition of its inputs.

Theorem 1: Let $\{\mathbf{S}_i^0\}_I$ be automata satisfying Cond. 1, and $(\{\mathbf{S}_i\}_I) = \text{NEGOTIATION}(\{\mathbf{S}_i^0\}_I)$. We then have that $\sup\mathcal{C}\big(\|_{i\in I}\, S_i\big) = \sup\mathcal{C}\big(\|_{i\in I}\, S_i^0\big)$, where $\sup\mathcal{C}$ is taken with respect to $\mathcal{L}(\|_{i\in I}\, \mathbf{M}_i)$ and $\bigcup_{i\in I}\Sigma_{i,\text{uc}}^{\text{p}}$. □

Proof: This proof is a generalisation from two to more components of the proof of Theorem 1 in [4]. To see that, just observe the definition of CSYNTH, Lem. 1, Rem. 1, and the following facts. First, note that components do not synchronise over their private events Σ_i^{p}, for all $i \in I$, and that here we assume any event shared event between at least two components is controllable by at least one of them (so in [4], $\Sigma_{\text{uc}}^{\text{s}} = \emptyset$). Finally, note that $\hat{\mathbf{S}}_i^k = \text{CSYNTH}(\mathbf{S}_i^{k-1})$ for all $i \in I$, for the following reasons (consider $\hat{\mathbf{S}}_i^k$ and $\mathbf{S}_i^k$ as in the proof of Lem. 1). For $k = 1$, we have that $\hat{\mathbf{S}}_i^1 \leftarrow \text{CSYNTH}(\mathbf{S}_i^0)$, as line **??** is executed for all $i \in I$. For $k > 1$, this line is not executed iff $\mathbf{S}_i$ did not receive any new contract in the previous round, or iff this exchange did not affect the semantics of $\mathbf{S}_i$, i.e., iff $\hat{\mathbf{S}}_i^{k-1} = \mathbf{S}_i^{k-1}$. ■

However, this does not yet constitute a solution to Problem 1, as NEGOTIATION may not remove enough to guarantee that the resulting composed system is nonblocking. This is only the case if the output of NEGOTIATION is nonconflicting, as formalised next.

Corollary 1: Let $\{\mathbf{S}_i^0\}_I$ be automata satisfying Cond. 1, and $(\{\mathbf{S}_i\}_I) = \text{NEGOTIATION}(\{\mathbf{S}_i^0\}_I)$. If $\{S_i\}_I$ are nonconflicting, i.e., if $\|_{i\in I}\ \mathbf{S}_i$ is nonblocking, we have that $\|_{i\in I}\, S_i = \sup\mathcal{C}\big(\|_{i\in I}\, S_i^0\big)$, where $\sup\mathcal{C}$ is taken with respect to $\mathcal{L}(\|_{i\in I}\, \mathbf{M}_i)$ and $\bigcup_{i\in I}\Sigma_{i,\text{uc}}^{\text{p}}$. □

Proof: This proof is a generalisation from two to more components of the proof of Corollary 1 in [4], if we note the definition of CSYNTH, and the following. Components do not synchronise over their private events Σ_i^{p}, for all $i \in I$, and here we assume any event shared event between at least two components is controllable by at least one of them (so in [4], $\Sigma_{\text{uc}}^{\text{s}} = \emptyset$). ■

By combining Lem. 1 and Cor. 1, we see that a solution to Problem 1 requires to additionally enforce nonconflict. This is achieved via ENFORCERU (line 12 in Proc. 1, and described in Sec. II-C) for particular classes of architectures, as discussed next.

VII. ADMISSIBLE ARCHITECTURES FOR NONCONFLICT

This section discusses how nonconflict can be enforced in a local, privacy-preserving manner for a class of admissible architectures, formalised in Sec. VII-A. Thereafter, we show how such architectures can be composed to larger admissible architectures in Sec. VII-B. If admissibility cannot be guaranteed, Sec. VII-C discusses how local coordinators can be incorporated to ensure nonconflict.

A. Admissible Architectures

We call a plant $\mathcal{M} = \{\mathbf{M}_i\}_I$ with the set of groups $\mathcal{G} = \{g_k\}_K$ admissible, if the following hypothesis holds.

Hypothesis 1: Let $\{\mathbf{S}_i^0\}_I$ be automata satisfying Cond. 1, and $(\{\mathbf{S}_i\}_I) = \text{NEGOTIATION}(\{\mathbf{S}_i^0\}_I)$. If, for all $i \in I$, and for all $g \in \mathcal{G}_i$, we have that $\{S_i\}_I$ is unambiguous with respect to Σ^g and the natural projection P_{ig}, then these languages are nonconflicting. □

In order to identify a subclass of admissible architectures, we use the following definition.

Definition 4: A *cycle of groups* is a set $\mathcal{G}' \subseteq \mathcal{G}$, let us say $\{g_k\}_{K'}$, with $|\mathcal{G}'| > 2$ and $K' \subseteq K$, such that there is $Q = \{1, \ldots, |\mathcal{G}'|\}$ and an isomorphism $o : K' \to Q$ where: (i) for all $k \in K'$, $\hat{g}_{o(k)} = g_k$, and (ii) $I^{\hat{g}_1} \cap I^{\hat{g}_m} \neq \emptyset$, and (iii) for all $1 \leq q < m$, we have that $I^{\hat{g}_q} \cap I^{\hat{g}_{q+1}} \neq \emptyset$. Denote by $\Sigma^{\mathcal{G}'} = \bigcup_{g\in\mathcal{G}'} \Sigma^g$. The cycle $\mathcal{G}'$ is a *minimal cycle of groups* if there is no subcycle $\mathcal{G}'' \subset \mathcal{G}'$.

With this, we define four different basic architectures.

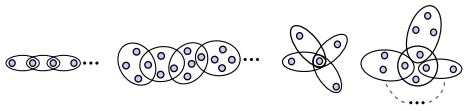

Fig. 2. Examples of linear (at the left) and star (at the right) architectures.

Definition 5: A group set $\mathcal{G}$ forms a *single-group architecture* if it only contains one group g, with $I^g = I$.

Definition 6: A set of groups $\mathcal{G} = \{g_k\}_K$ forms a *linear architecture* if there exists an isomorphism $o : K \to K$ such that: (i) for all $k \in K$, $\hat{g}_{o(k)} = g_k$, and (ii) $I^{\hat{g}_1} \cap I^{\hat{g}_m} = \emptyset$, and (iii) for all $1 < q < m$ and for all $q' \in K$, $q' \neq q$, we have that $I^{\hat{g}_q} \cap I^{\hat{g}_{q'}} \neq \emptyset$ if and only if $q' = q + 1$ or $q' = q - 1$.

Definition 7: A set of groups $\mathcal{G} = \{g_k\}_K$ forms a *star architecture* if there is $g \in \mathcal{G}$ where, for all $g', g'' \in \mathcal{G}\backslash\{g\}$, $I^g \cap I^{g'} \neq \emptyset$ and $I^{g'} \cap I^{g''} = \emptyset$.

Definition 8: A set of groups $\mathcal{G} = \{g_k\}_K$ forms a *cyclic-with-lookout architecture* if $\mathcal{G}$ is a minimal cycle of groups, as in Def. 4, and there is a *lookout* component $i \in I$ such that $\Sigma^{\mathcal{G}} \subseteq \Sigma_i$.

Indeed, as the main result of this section, we now show that the three architectures from Def. 5-8 are admissible.

Proposition 1: For a system $\mathcal{M}$ that has either a single-group, a linear, a star or a cyclic-with-lookout architecture, Hyp. 1 is valid. □

Proof: For the single-group architecture, this is a trivial extension of Proposition 1 in [4]. For the other architectures, note that the following statements hold:

1. $\forall i \in I.\ \forall g \in \mathcal{G}_i$: (i) $\mathbf{S}_i$ is trim, (ii) $\forall j \in I^g.\ P_{ig}S_i = P_{jg}S_j$, and (iii) S_i is unambiguous with respect to Σ^g.

To prove nonconflict, we have to prove that $\|_{i\in I}\ \overline{S_i} \subseteq \overline{\|_{i\in I}\ S_i}$. Take any $t \in \|_{i\in I}\ \overline{S_i}$. Denote $t_i = P_i t$. Let us fix an $i \in I$. Note that there is $u_i \in \Sigma_i^*$ such that $t_i u_i \in S_i$. Then, because of 1.(ii), we have that, for all $g \in \mathcal{G}_i$ and all $j \in I^g$, there is $\hat{t}_j\hat{u}_j \in S_j$ such that $P_{jg}\hat{t}_j = P_{ig}t_i$, and $P_{jg}\hat{u}_j = P_{ig}u_i$. In addition, because of 1.(iii), and since $\hat{t}_j\hat{u}_j \in S_j$, there is u_j such that $t_j u_j \in S_j$ (see the details on

how to construct this u_j using unambiguity from the proof of Proposition 1 in [4]).

Now let us first finish the proof for the linear and star architectures. For them, as a consequence of the definition of groups, note the following also holds:

2. (i) $\forall i \in I.\ \forall g_1, g_2 \in \mathcal{G}_i$: $g_1 \neq g_2 \rightarrow \Sigma^{g_1} \cap \Sigma^{g_1} = \emptyset$, and (ii) there is no cycle of groups.

From 2.(i) we know, for all $j_1 \in I^{g_1}$ and all $j_2 \in I^{g_2}$, with $j_1 \neq j_2$, that $t_{j_1}u_{j_1}$ and $t_{j_2}u_{j_2}$ do not have any events in common, i.e., do not synchronise, so $P_{j_1}^{-1}u_{j_1} \cap P_{j_2}^{-1}u_{j_2} \neq \emptyset$. Moreover, because of 2.(ii), we can inductively apply the same arguments from the previous paragraph to obtain, for all $l \in I$, a word u_l such that $t_l u_l \in S_l$ (first, for all components $p \in I$ that are in the same group as j, then, for all components $q \in I$ that share a group with p, and so on). Therefore, thanks to 2., we have that $\bigcap_{i\in I} P_i^{-1}u_i \neq \emptyset$. Take u from this intersection. Then, $tu \in \overline{\|_{i\in I} S_i}$.

Finally, let us finish the proof for the cyclic-with-lookout architecture. Note that, in principle, when we have a cycle, we cannot apply the same inductive arguments used above to obtain u. That is because eventually we reach the same component in the cycle, let us say k, coming from different directions of induction. That is, following different sequences of groups from component i to k during the induction, we cannot guarantee there is u_k such that $\bigcap_{i\in I} P_i^{-1}u_i \neq \emptyset$. However, in the presence of a lookout component, let us say c, we can do it. The reason is that, from component i, we can choose c as the first step of the induction (as c, being a lookout, shares events with i). Then, u_c serves as a template that imposes a feasible sequence between all shared events in the cycle, $\Sigma^{\mathcal{G}}$, guaranteeing that we can follow the induction such that $\bigcap_{i\in I} P_i^{-1}u_i \neq \emptyset$. ■

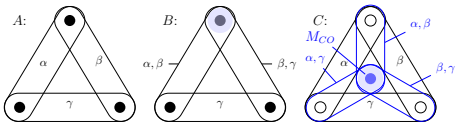

Fig. 3. Cyclic architectures without (A) and with lookout (B and C).

B. Connecting Admissible Architectures.

Definition 9: Let $\{\mathcal{M}_u\}_U$ be a set of systems, where U is a set of indexes. Assume that, for each $u \in U$, $\mathcal{M}_u$ is either a single-group, a linear, a star or a cyclic-with-lookout architecture. For different $u, u' \in U$, consider the systems may have common components, i.e., $\mathcal{M}_u \cap \mathcal{M}'_u \neq \emptyset$. We then say the system $\mathcal{M} = \bigcup_{u\in U} \mathcal{M}_u$ has a *mixed architecture*.

Proposition 2: For a system $\mathcal{M}$ with a mixed architecture, Hyp. 1 is valid if there is no set $\mathcal{G}' \subseteq \mathcal{G}$ such that $\mathcal{G}'$ is a minimal cycle without lookout, i.e., if for all $\mathcal{G}' \subseteq \mathcal{G}$, $\mathcal{G}'$ being a minimal cycle implies that there is $i \in \bigcup_{g\in\mathcal{G}'} I^g$ such that $\Sigma^{\mathcal{G}'} \subseteq \Sigma_i$. Then $\mathcal{M}$ is also admissible. □

Proof: For this proof, we can use the same steps as in the proof of Prop. 1, if we observe the assumption that the connection between the systems $\mathcal{M}_u$ does not close any new cycle, apart from already existing ones in plants $\mathcal{M}_u$ that have a cyclic-with-lookout architecture. ■

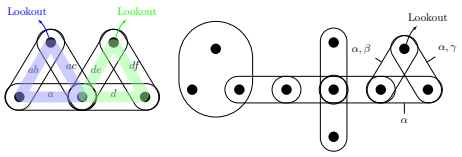

Fig. 4. Examples of admissible mixed architectures.

C. Coordinators for Non-Admissible Architectures.

Unfortunately, there are very simple architectures for which admissibility cannot be guaranteed.

Definition 10: A set of groups $\mathcal{G} = \{g_k\}_K$ forms a *cyclic-without-lookout architecture* if $\mathcal{G}$ is a minimal cycle of groups, as in Def. 4, but there is no lookout component, i.e., there is no $i \in I$ such that $\Sigma^{\mathcal{G}} \subseteq \Sigma_i$.

Fig. 5 shows an example of a system with a the cyclic-without-lookout architecture from Fig. 3 (A) for which Hyp. 1 is not valid.

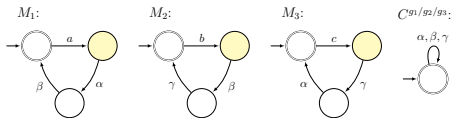

Fig. 5. Example of system whose architecture requires a coordinator.

In order to still guarantee nonconflict via CBSS, a local coordinator can be obtained in the following manner. We can convert this architecture into a cyclic-with-lookout one, which is admissible, by adding a component $\mathbf{M}_{\text{co}}$ in the plant $\mathcal{M}$, as follows. Define $\mathbf{M}_{\text{co}}$ as the single state automaton that accepts and generates $\Sigma^{\mathcal{G}*}$, such that it acts as a lookout for the shared events in the cycle, i.e., $\Sigma^{\mathcal{G}}$. In this new architecture (see Fig. 3, (C)) we can guarantee nonconflict, as per Prop. 1, and $\mathbf{S}_{\text{co}}$ is then a coordinator for the components in the cycle.

VIII. Decentralised Supervisory Control

The goal of CBSS is to compute automata $\mathbf{S}_i$ which allow to extract supervisors f_i which solve Problem 1. We first discuss in Sec. VIII-A how cooperative supervisors can be extracted from the output of CBSS, and then show in Sec. VIII-B that these supervisors indeed solve Problem 1.

A. Extraction of Cooperative Supervisors

As any local closed loop S_i in the output of CBSS ultimately results from CSYNTH (Def. 1), we know that local controllability of S_i is respected in terms of $\Sigma^{\text{p}}_{i,\text{uc}}$. Unfortunately, this does not imply that S_i is controllable with respect to $\Sigma^{\text{s}}_{i,\text{cp}}$. Thus, we cannot simply take f_i as in Sec. II-B, that is, such that $\mathbf{S}_i \sim f_i$. To illustrate that, let us say that, in order to respect local controllability, we extract a local supervisor $f_i : \mathcal{L}(\mathbf{M}_i) \rightarrow \Gamma_i$ which only disables locally controllable events in the set $\Sigma_{i,\text{c}}$, that is, for all $s \in \mathcal{L}(\mathbf{M}_i)$, define $f_i(s) = \{\sigma \in \Sigma_i \mid s\sigma \in \overline{S_i}\} \cup \Sigma^{\text{s}}_{i,\text{cp}} \subseteq \Sigma_i$. In this case, the simple example depicted in Fig. 6 illustrates why cooperation may fail: supervisor i expects cooperation from other components in a group g to disable, at the state 0, the event $\theta \in \Sigma^{\text{s}}_{i,\text{cp}}$; however, this is not expressed in the

contract $\mathbf{C}_i^g$, because i needs θ to be enabled at state 1, while states 0 and 1 correspond to the same state in the contract, and therefore are indistinguishable to j.

Fig. 6. Cooperation might fail in the absence of cooperative messages.

To solve this problem, we follow [4] and consider the use of a signalling bit for each $\sigma \in \bigcup_{i\in I} \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$. A supervisor i that does not control σ can request who controls it to disable it on i's behalf. The request is done by setting the bit relative to σ to one, which is represented by $\mathrm{d}^\sigma \in \Sigma^{\mathrm{d}}_{i\to}$, as defined in Sec. II-C. We model the requests made by i to other members it shares a group with by a function $d_{i\to} : \mathcal{L}(\mathbf{M}_i) \to 2^{\Sigma^{\mathrm{d}}_{i\to}}$. For all $w_i \in \mathcal{L}(\mathbf{M}_i)$ and all $\mathrm{d}^\sigma \in \Sigma^{\mathrm{d}}_{i\to}$, we interpret this function such that $\mathrm{d}^\sigma \in d_{i\to}(w_i)$ if and only if supervisor i assigns one to the bit relative to $\sigma \in \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$ after observing w_i. We assume this to be instantaneous. That is, we assume, for all $\alpha \in \Sigma_i$, that the output $d_{i\to} = d_{i\to}(w_i)$ is immediately updated to $d_{i\to} = d_{i\to}(w_i\alpha)$ with the occurrence of α. We then define $d_{i\to}(w_i) = \{\mathrm{d}^\sigma \,|\, w_i\sigma \notin \overline{S_i}\}$. If multiple supervisors share σ but do not control it, they all have access to write on its signalling bit. In order to achieve joint control, we assume the value of this bit is a logical *or* function of the value each supervisor tries to assign to it. Therefore, for each group $g \in \mathcal{G}$, we define $d_g = \bigcup_{i\in I^g} d_{i\to}$. Finally, the set of requests *read* by i from the signalling bits – relative to events controlled by i and shared with other supervisors – is $d_{\to i} = \Sigma^{\mathrm{d}}_{\to i} \cap \bigcup_{g\in\mathcal{G}_i} d_g$. We then extract a supervisor as follows, which allows us to state Lem. 2 below.

Definition 11: Let $\{\mathbf{S}_i^0\}_I$ be automata satisfying Cond. 1, and $(\{\mathbf{S}_i\}_I) = \textsc{Negotiation}(\{\mathbf{S}_i^0\}_I)$. If $\mathbf{S}_i$ are nonempty automata, we define each local supervisor $f_i : \mathcal{L}(\mathbf{M}_i) \times 2^{\Sigma^{\mathrm{d}}_{\to i}} \to \Gamma_i$, with $\Gamma_i \subseteq 2^{\Sigma_i}$, such that for all $w_i \in \mathcal{L}(\mathbf{M}_i)$ and for all $d_{\to i} \in 2^{\Sigma^{\mathrm{d}}_{\to i}}$, we have $f_i(w_i, d_{\to i}) = \{\sigma \,|\, w_i\sigma \in \overline{S_i}$ and $(\sigma \in \Sigma^{\mathrm{s}}_{i,\mathrm{c}} \to \mathrm{d}^\sigma \notin d_{\to i})\} \cup \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$.

Lemma 2: In the context of Def. 11, we have that

1) $\|_{i\in I}\, \mathcal{L}_m(f_i/\mathbf{M}_i) = \|_{i\in I}\, S_i$ and
2) $\|_{i\in I}\, \mathcal{L}(f_i/\mathbf{M}_i) = \|_{i\in I}\, \overline{S_i}$.

Proof: This proof is a generalisation from two to more components of the proof of Lemma 2 in [4]. ■

From this lemma, the next result immediately holds.

Corollary 2: The plant $\mathcal{M} = \{\mathbf{M}_i\}_I$ in closed loop with the supervisors f_i from Def. 11 is nonblocking, that is, $\|_{i\in I}\, \mathcal{L}(f_i/\mathbf{M}_i) = \overline{\|_{i\in I}\, \mathcal{L}_m(f_i/\mathbf{M}_i)}$, if and only if $\{S_i\}_I$ are nonconflicting, i.e., if and only if $\|_{i\in I}\, \mathbf{S}_i$ is nonblocking. □

B. Solving Problem 1

By combining multiple previous results, we have the following soundness result of Proc. 1, which shows that, if found, the fully local supervisors extracted from the output of CBSS indeed solve Problem 1.

Theorem 2: Consider a plant $\mathcal{M}$ that has an admissible architecture. Let $\{\mathbf{K}_i\}_I$ be the set of its plantified specifications, and $(\{\mathbf{S}_i\}_I) = \mathrm{CBSS}(\{\mathbf{K}_i\}_I)$. If there is no $i \in I$ such that $\mathbf{S}_i$ is the empty automaton, let f_i be the local supervisors defined from $\mathbf{S}_i$ via Def. 11. Then, it holds that

1) $\|_{i\in I}\, \mathcal{L}_m(f_i/\mathbf{M}_i) \in \mathcal{C}(\|_{i\in I}\, K_i)$, and
2) $\|_{i\in I}\, \mathcal{L}(f_i/\mathbf{M}_i) = \overline{\|_{i\in I}\, \mathcal{L}_m(f_i/\mathbf{M}_i)}$,

where controllability is taken with respect to $\mathcal{L}(\|_{i\in I}\, \mathbf{M}_i)$ and $\bigcup_{i\in I} \Sigma^{\mathrm{p}}_{i,\mathrm{uc}}$. □

Proof: Note the following two facts, so we can apply previous results from this paper. Firstly, the output of CBSS is also the output of its last call of NEGOTIATION. Secondly, each $\mathbf{S}_i$ outputted by CBSS satisfies Cond. 1, because: $\mathbf{K}_i$ satisfies Cond. 1, $\mathbf{S}_i$ outputted by NEGOTIATION also satisfies it (see Lem. 1), and by Remark 7 from [4], the output of ENFORCERU also satisfies it. Thanks to ENFORCERU ([4]), for all $i \in I$ we have that $\mathbf{S}_i$ outputted by CBSS is unambiguous with respect to Σ_i^g, for all $g \in \mathcal{G}_i$. Moreover, $\mathbf{S}_i$ is trim. Hence, by Prop. 1 and 2, we get that S_i are nonconflicting. This proves (2) by Cor. 2. Moreover, from nonconflict, by Cor. 1 we get that $\|_{i\in I}\, S_i = \mathrm{sup}\mathcal{C}(\|_{i\in I}\, \tilde{S}_i)$, where $\tilde{\mathbf{S}}_i$ is an automaton that satisfies Cond. 1 and such that $\tilde{S}_i \subseteq K_i$. Thus, we have that $\mathrm{sup}\mathcal{C}(\|_{i\in I}\, \tilde{S}_i) \subseteq \mathrm{sup}\mathcal{C}(\|_{i\in I}\, K_i)$, so $\|_{i\in I}\, S_i \in \mathcal{C}(\|_{i\in I}\, K_i)$. Finally, we can apply Lem. 2, which proves (1). ■

REFERENCES

[1] A. Sangiovanni-Vincentelli, W. Damm, and R. Passerone, "Taming dr. frankenstein: Contract-based design for cyber-physical systems," *European journal of control*, vol. 18, no. 3, pp. 217–238, 2012.

[2] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, "Contracts for system design," *Foundations and Trends in Electronic Design Automation*, vol. 12, no. 2-3, pp. 124–400, 2018.

[3] A. M. Mainhardt and A.-K. Schmuck, "Assume-guarantee synthesis of decentralised supervisory control," *IFAC-PapersOnLine*, vol. 55, no. 28, pp. 165–172, 2022, 16th IFAC Workshop on Discrete Event Systems WODES 2022.

[4] A. Mainhardt and A.-K. Schmuck, "Synthesis of decentralized supervisory control via contract negotiation," *IEEE Transactions on Automatic Control.*, 2025, (to appear), Preprint available here.

[5] R. Su and J. Thistle, "A distributed supervisor synthesis approach based on weak bisimulation," in *2006 8th International Workshop on Discrete Event Systems*, 2006, pp. 64–69.

[6] K. Cai and W. M. Wonham, *Supervisor Localization, A Top-Down Approach to Distributed Control of Discrete-Event Systems*, ser. Lecture Notes in Control and Information Sciences. Springer, 2016.

[7] J. Komenda and T. Masopust, "Computation of controllable and coobservable sublanguages in decentralized supervisory control via communication," *Discrete Event Dynamic Systems*, vol. 27, 12 2017.

[8] W. M. Wonham and K. Cai, *Supervisory control of discrete-event systems*, ser. Communications and Control Engineering. Springer, 2019.

[9] H. Flordal, R. Malik, M. Fabian, and K. Åkesson, "Compositional synthesis of maximally permissive supervisors using supervision equivalence," *Discret. Event Dyn. Syst.*, vol. 17, no. 4, pp. 475–504, 2007.

[10] W. A. Apaza-Perez, C. Combastel, and A. Zolghadri, "On distributed symbolic control of interconnected systems under persistency specifications," *International Journal of Applied Mathematics and Computer Science*, vol. 30, no. 4, 2020.

[11] R. Majumdar, K. Mallik, A. K. Schmuck, and D. Zufferey, "Assume-guarantee distributed synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3215–3226, 2020.

[12] B. Finkbeiner and N. Passing, "Compositional synthesis of modular systems," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2021, pp. 303–319.

[13] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 3rd ed. Springer, 2021.