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 components by forcing them to locally satisfy *compatible contracts* in order to achieve a desired global behaviour. The advantages are threefold: (i) decentralised design (controllers can be synthesised and executed locally); (ii) information privacy, as no detailed information about a component's behaviour is shared with the rest of the system, apart from what is specified by the contracts; and (iii) decoupled maintenance (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, components of a system synchronise via shared events and are controlled by supervisors *local* to each one of these components. In this context, *compatible contracts* model required behavioural restrictions of sets of components solely in terms of events they share. Our previous work [3], [4] introduced a two-component CBSS approach for cosynthesising such compatible contracts and their enforcing local supervisors. In this paper, we generalise this approach to multicomponent architectures. This introduces new challenges for privacy preservation, as synchronisation occurs between 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 components, is a critical step in decentralised DES. In particular for CBSS, where supervisors are designed locally and do not observe private events of other components they synchronise with. 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. This is done 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 components, 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 intervehicle 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.

II. Preliminaries

A. Languages and Automata Basics

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

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 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 $A = \{a_1, ..., a_n\}$, we write $A = \{a_i\}_I$ for $I = \{1, ..., n\}$. If a_i is any constant value a for all i, we write $A = \{a\}_I$. If $A = \{(a_1, b_1, ...), ..., (a_n, b_n, ...)\}$, we write $A = \{(a_i, b_i, ...)\}_I$.
Languages. A *language* over Σ is a subset $L \subseteq \Sigma^*$. The *prefix* of a language $L \subseteq \Sigma^*$ is defined by $\overline{L} := \{r \in \Sigma^* \,|\, \exists s \in L : r \leq s\}$. 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 $\delta(\delta(q, s), \sigma))!$. Unless $\mathbf{\Lambda}$ is what is termed *the empty automaton* ([13]), where $Q = \emptyset$ and q_0 is not defined, we say $\mathbf{\Lambda}$ is *nonempty*.
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* $\mathcal{L}(\mathbf{\Lambda}) := \{\, s \in \Sigma^* \,|\, \delta(q_0, s)! \,\}$ and the *marked language* $\mathcal{L}_m(\mathbf{\Lambda}) := \{\, s \in \Sigma^* \,|\, \delta(q_0, s) \in Q_\mathrm{m} \,\}$. 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. That is, 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 such that it generates and accepts the languages $P_\mathrm{s}(\mathcal{L}(\mathbf{\Lambda}))$ and $P_\mathrm{s}(\Lambda)$, respectively, where $P_\mathrm{s} : \Sigma^* \to \Sigma^{\mathrm{s}*}$ is a natural projection. 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 – which cannot be disabled.
Plantified specification. For any prefix-closed specification language E over any $\Sigma' \subseteq \Sigma$, 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 $\mathcal{C}(L) := \{L' \subseteq L \,|\, L'$ is controllable with respect to $\mathcal{L}(\mathbf{M})$ 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 $\Gamma := \{\gamma \subseteq \Sigma \,|\, \Sigma_\mathrm{uc} \subseteq \gamma\} \subseteq 2^\Sigma$ 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 $\mathcal{L}(f/\mathbf{M}) \subseteq \mathcal{L}(\mathbf{M})$. 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 $\Sigma := \Sigma_{\rm c}\dot{\cup}\Sigma_{\rm uc}$, the control problem is to synthesise the *maximally permissive* supervisor f that: (i) 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 $\Sigma_{\rm uc}$ – and (ii) 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 $\Sigma = \bigcup_{i\in I}\Sigma_i$. 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 $\Sigma_i^{\rm s} = \bigcup_{j\in I, j\neq i}\Sigma_{i,j}$. A local alphabet is partitioned into $\Sigma_i = \Sigma_i^{\rm p}\dot{\cup}\Sigma_i^{\rm s}$, where $\Sigma_i^{\rm p}$ is the set of *private* events of i. We assume that $\Sigma_i^{\rm p}$ may contain controllable and uncontrollable events, meaning $\Sigma_i^{\rm p} = \Sigma_{i,\rm c}^{\rm p}\dot{\cup}\Sigma_{i,\rm uc}^{\rm p}$. We denote by P_{is} and P_i the natural projections $P_{is} : \Sigma_i^* \to \Sigma_i^{{\rm s}*}$ 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 $\sigma \in \Sigma_i^{\rm s}$, there is j (possibly $j = i$) such that σ is controllable by j. Thus, we have that $\Sigma_i^{\rm s} = \Sigma_{i,\rm c}^{\rm s}\dot{\cup}\Sigma_{i,\rm cp}^{\rm s}$, where $\Sigma_{i,\rm c}^{\rm s}$ are shared events controllable by i, and $\Sigma_{i,\rm cp}^{\rm s}$ 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 $\Sigma_{i,\rm cp}^{\rm s}$ 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 $\Sigma_{i,\rm uc}^{\rm p}$, instead of the actual locally uncontrollable event set $\Sigma_{i,\rm uc} = \Sigma_{i,\rm uc}^{\rm p}\dot{\cup}\Sigma_{i,\rm cp}^{\rm s}$.

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

Definition 1: For any automaton $\mathbf{\Lambda}_i$ over Σ_i, the *cooperative* synthesis function CSYNTH is defined such that CSYNTH$(\mathbf{\Lambda}_i)$ is a trim automaton accepting the language $\sup\mathcal{C}(\Lambda_i)$ with respect to $\mathbf{\Lambda}_i$ and $\Sigma_{i,\rm uc}^{\rm p}$. □

Remark 1: Let $\mathbf{\Lambda}_i$ be an automaton that satisfies Cond. 1 (e.g., the plantified specification $\mathbf{K}_i$). Then $\sup\mathcal{C}(\Lambda_i)$ with respect to $\mathbf{M}_i$ and $\Sigma_{i,\rm uc}^{\rm p}$ is the same as with respect to $\mathbf{\Lambda}_i$ and $\Sigma_{i,\rm uc}^{\rm p}$. (i.e., such $\mathbf{\Lambda}_i$ carries enough information for controllability to be checked, without the need to manipulate $\mathbf{M}_i$ in addition). Moreover, CSYNTH$(\mathbf{\Lambda}_i)$ 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]): A language Λ_i over Σ_i is *unambiguous* with respect to any $\Sigma^g \subseteq \Sigma_i$ if, for any $s, s' \in \overline{S_i}$ where $P_{ig}(s) = P_{ig}(s')$ (projection $P_{ig} : \Sigma_i^* \to \Sigma^{g*}$), 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 $\mathbf{\Lambda}_i$ satisfying Cond. 1, the automaton ENFORCERU$(\mathbf{\Lambda}_i, \Sigma^g)$ also satisfies Cond. 1, and its accepted language is unambiguous with respect to Σ^g. Moreover, if $\mathbf{\Lambda}_i$ accepted language is already unambiguous with respect to Σ^g, then $\mathbf{\Lambda}_i \equiv$ ENFORCERU$(\mathbf{\Lambda}_i, \Sigma^g)$. □

Signalling Bits. In order to extract from $\mathbf{S}_i$ local supervisors f_i that actually exhibit the intended cooperation whenever required (i.e. whenever $\Sigma_{i,\rm cp}^{\rm s} \neq \emptyset$) we propose, as in [4], the use of a signalling bit for each shared event $\sigma \in \bigcup_{i\in I}\Sigma_{i,\rm cp}^{\rm s}$. We define $\Sigma^{\rm d} = \{{\rm d}^\sigma \mid \sigma \in \Sigma_{i,\rm cp}^{\rm s}$ and $i \in I\}$, where ${\rm d}^\sigma$ 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 $\Sigma_{i\to}^{\rm d} = \{{\rm d}^\sigma \mid \sigma \in \Sigma_{i,\rm cp}^{\rm s}\}$ and $\Sigma_{\to i}^{\rm d} = \Sigma^{\rm d} \cap \{{\rm d}^\sigma \mid \sigma \in \Sigma_{i,\rm c}^{\rm s}\}$ 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 $f_i : \mathcal{L}(\mathbf{M}_i) \times 2^{\Sigma_{\to i}^{\rm d}} \to \Gamma_i$ 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.

[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.

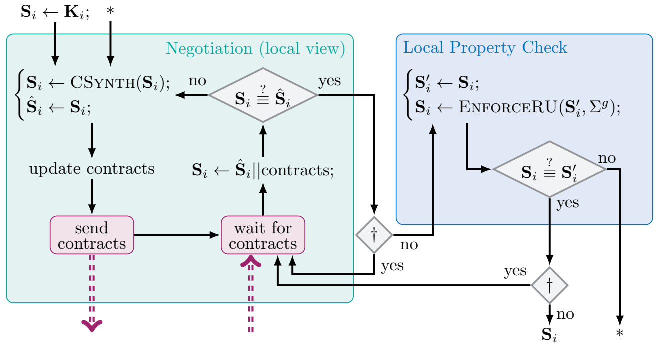

Fig. 1. Simplified *local overview* of CBSS for component i. Functions CSYNTH and ENFORCERU are as in Def. 1 and Rem. 2, respectively. Symbol † stands for "Have other subsystems updated their contracts?". Synthesis is initialised with plantified specifications $\mathbf{K}_i$, and terminates when all 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], from two to multicomponent architectures. What distinguishes CBSS from other solutions to Problem 1 is its strong emphasis on privacy: both during the synthesis and the execution of the supervisors, components observe solely the *shared events* of other components. In order to still remain permissive, CBSS iteratively refines *contracts* that represent cooperative *agreements*.

Contract Negotiation. In distributed architectures, distinct set of events, let us say Σ^g, are shared among distinct groups g of components (as formally defined in Sec. IV). So, in contrast to [3], [4], each component i negotiates individual contracts $\mathbf{C}_i^g$, one for each group g that i belongs to. The behaviour of a component in terms of its private events (with respect to each group) should remain private. Thus, we define $\mathbf{C}_i^g := \mathbf{O}_{\mathbf{S}_i,\Sigma^g}$[2], where $\mathbf{S}_i$ is the automaton used to extract the local supervisor f_i. 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 $\mathbf{S}_i$ via CSYNTH (Def. 1), update their contracts $\mathbf{C}_i^g$, and negotiate these contracts within each group, as illustrated by the *Negotiation* box in Fig. 1. These iterations end 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. In contrast, only local specifications were allowed in [3], [4]. Moreover, in Sec. VII-B we show how coordinators can be obtained under our CBSS approach in systems with architectures that require partially 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 5, we illustrate few possible architectures of plants based on the groups they can form.

Definition 3 (Groups): For a plant $\mathcal{M} = \{\mathbf{M}_i\}_I$, and for any $g = (I^g, \Sigma^g) \in 2^I \times (2^\Sigma \backslash \{\emptyset\})$, 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 $\Sigma_{i,j} \neq \Sigma^g$. Denote by $\mathcal{G}$ the set of all different groups g for the plant $\mathcal{M}$. For all $i \in I$, denote by $\mathcal{G}_i = \{g \in \mathcal{G} \,|\, i \in I^g\}$ the set of all groups to which i belongs, and for all $g \in \mathcal{G}_i$, define the projections $P_{ig} : \Sigma_i^* \to \Sigma^{g*}$.

A. Plants with Nonlocal Specifications

Within the framework presented here, we can also impose nonlocal specifications, which are not 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 that cannot be decomposed over local alphabets Σ_i. Let us say E over an alphabet $\Sigma' \subseteq \Sigma$. 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 set $\mathcal{M}$, as follows. Define $\mathbf{M}_E$ as the single state automaton that accepts and generates Σ'^*, so to guarantee controllability is respected. Then, from $\mathbf{M}_E$ and $\mathbf{E}$ we can obtain a plantified specification $\mathbf{K}_E$ according to Sec. II-B.

[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] 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 (also possibly multiple) groups g.

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 (Def. 1). 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 non-equivalent to the input $\mathbf{S}_i^0$.

Procedure 1 NEGOTIATION

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

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 POP. 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. This is done 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 2 CBSS

Require: Plantified-specification automata $\{\mathbf{K}_i\}_I$.
1: $WhereToEnforceRU \leftarrow \{(i,g) \in I \times \mathcal{G} \mid i \in g\}$
2: $\{(\mathbf{S}_i, changed_i, synth_i)\}_I \leftarrow \{(\mathbf{K}_i, \mathtt{false}, \mathtt{true})\}_I$
3: **repeat**
4: $\quad \{(\mathbf{S}_i, changed_i)\}_I \leftarrow \text{NEGOTIATION}(\{\mathbf{S}_i\}_I, \{synth_i\}_I)$
5: $\quad \{synth_i\}_I \leftarrow \{\mathtt{false}\}_I$
6: $\quad$ **if** $\exists i$ such that $\mathbf{S}_i$ is the empty automaton **then go to** line 17
7: $\quad$ **for** $(i,g) \in \{(i,g) \in I \times \mathcal{G} \mid i \in g \text{ and } changed_i\}$ **do**
8: $\quad\quad WhereToEnforceRU \leftarrow WhereToEnforceRU \cup \{(i,g)\}$
9: $\quad enforcedOnOne \leftarrow \mathtt{false}$, $k \leftarrow -1$
10: $\quad$ **while** $WhereToEnforceRU \neq \emptyset$ and $\neg enforcedOnOne$ **do**
11: $\quad\quad (k,h) \leftarrow \text{POP}(WhereToEnforceRU)$
12: $\quad\quad (\mathbf{S}_k, enforcedOnOne) \leftarrow \text{ENFORCERU}(\mathbf{S}_k, \Sigma^h)$
13: $\quad\quad$ **if** $enforcedOnOne$ **then** $synth_k \leftarrow \mathtt{true}$
14: $\quad$ **if** $\exists i$ such that $\mathbf{S}_i$ is the empty automaton **then go to** line 17
15: $\quad$ **if** $\neg enforcedOnOne$ **then** $k \leftarrow -1$
16: **until** $k = -1$
17: **return** $\{\mathbf{S}_i\}_I$

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. 1 with $\{\mathbf{S}_i^0\}_I$ as its inputs, namely, $(\{\mathbf{S}_i\}_I) = \text{NEGOTIATION}(\{\mathbf{S}_i^0\}_I)$. 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 $i \in I$. $\square$

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 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 22 in NEGOTIATION, for $k \geq 1$. Given the initial value of the boolean variables in the procedure, for $k = 1$ 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 $\mathbf{S}_i^k = \hat{\mathbf{S}}_i^k \parallel \big(\|_{g \in \mathcal{G}_i} \|_{j \in I^g} \mathbf{O}_{\hat{\mathbf{S}}_j^k,\Sigma^g} \big)$ for all $i \in I$. 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 already incorporates that contract, if $\hat{\mathbf{S}}_j^k = \hat{\mathbf{S}}_j^{k-1}$ (i.e., if it did not change from the previous round). Finally, the fix point is reached when $\hat{\mathbf{S}}_i^k = \mathbf{S}_i^k$ holds for some $k > 1$ and for all $i \in I$. Then we have, for all $i, j \in I$ and all $g \in \mathcal{G}_i \cap \mathcal{G}_j$, 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 $i, j \in I$, we have that $P_{jg}\mathcal{L}(\mathbf{S}_j) = P_{ig}\mathcal{L}(\mathbf{S}_i)$ and $P_{jg}S_j = P_{ig}S_i$. $\blacksquare$

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 $(\{\mathbf{S}_i\}_I) = \textsc{Negotiation}(\{\mathbf{S}_i^0\}_I)$. 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 $\bigcup_{i\in I}\Sigma_{i,\mathrm{uc}}^{\mathrm{p}}$. □

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 between at least two components is controllable by at least one of them (so in [4], $\Sigma_{\mathrm{uc}}^{\mathrm{s}} = \emptyset$). Finally, note that $\hat{\mathbf{S}}_i^k = \textsc{CSynth}(\mathbf{S}_i^{k-1})$ 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 $\hat{\mathbf{S}}_i^1 \leftarrow \textsc{CSynth}(\mathbf{S}_i^0)$, as line 10 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 $\mathbf{S}_i^{k-1} = \hat{\mathbf{S}}_i^{k-1}$. ■

However, this does not yet constitute a solution to Problem 1, as NEGOTIATION may not refine $\mathbf{S}_i$ 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 $(\{\mathbf{S}_i\}_I) = \textsc{Negotiation}(\{\mathbf{S}_i^0\}_I)$. 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 $\bigcup_{i\in I}\Sigma_{i,\mathrm{uc}}^{\mathrm{p}}$. □

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

From Cor. 1 we see that a solution to Problem 1 still requires to enforce nonconflict. This is achieved via ENFORCERU (line 12 in Proc. 2, 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. If admissibility cannot be guaranteed, Sec. VII-B discusses how local coordinators can be incorporated to ensure nonconflict. Thereafter, in Sec. VII-C we show how such architectures can be composed to larger admissible architectures.

A. Admissible Architectures

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

Hypothesis 1: Let $\{\mathbf{S}_i^0\}_I$ be automata satisfying Cond. 1, and $(\{\mathbf{S}_i\}_I) = \textsc{Negotiation}(\{\mathbf{S}_i^0\}_I)$. 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. □

With this, we define four different basic architectures.

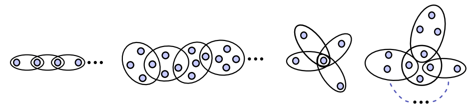

Fig. 2. Examples of linear (at the left) and star (at the right) architectures.

Definition 4: The group set $\mathcal{G}$ forms a *single-group architecture* if it only contains one group g, with $I^g = I$.

Definition 5: A set of groups $\mathcal{G}$ forms a *linear architecture* (depicted in Fig. 2) if its elements can be indexed as follows. We can write $\mathcal{G} = \{g_u\}_U$, with $U = \{1, \dots, m\}$ as the set of indexes and $2 \le m = |\mathcal{G}|$, such that: (i) $I^{g_1} \cap I^{g_m} = \emptyset$, and (iii) for all $q \in U\backslash\{1\}$ and for all $q' \in U$, $q' < q$, we have that $I^{g_q} \cap I^{g_{q'}} \neq \emptyset$ if and only if $q' = q - 1$.

Definition 6: The set of groups $\mathcal{G}$ forms a *star architecture* (illustrated in Fig. 2) 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 $(I^{g'} \cap I^{g''})\backslash I^g = \emptyset$.

In order to identify the next basic admissible architecture, we use the following definition.

Definition 7 (Cycle): A *cycle* is a set of groups $\mathfrak{C} \subseteq \mathcal{G}$ if its elements can be indexed as follows. We can write $\mathfrak{C} = \{g_u\}_U$, with $U = \{1, \dots, m\}$ as the set of indexes, and $2 < m = |\mathfrak{C}| \le |\mathcal{G}|$, such that: (i) $I^{g_1} \cap I^{g_m} \neq \emptyset$, and (ii) for all $1 \le v < m$, we have that $I^{g_v} \cap I^{g_{v+1}} \neq \emptyset$. The cycle $\mathfrak{C}$ is *minimal* if there is no cycle $\mathfrak{C}' \subset \mathfrak{C}$.

Definition 8: The set of groups $\mathcal{G}$ forms a *cyclic-with-lookout architecture* if $\mathcal{G}$ is the union of minimal cycles, and if there is a *lookout* component $i \in I$ such that $\Sigma^{\mathcal{G}} = \bigcup_{g\in\mathcal{G}}\Sigma^g = \bigcup_{j\in I}\Sigma_j^{\mathrm{s}} \subseteq \Sigma_i$. See examples in Fig. 3 (B, C).

Indeed, as the main result of this section, we now show that the three architectures given above 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 $t \in \|_{i\in I}\, \overline{S_i}$, and denote $t_i = P_i t$ for any $i \in I$. Let us fix an arbitrary $l \in I$. Note that there is $u_l \in \Sigma_l^*$ such that $t_l u_l \in S_l$. Then, for all $g \in \mathcal{G}_l$ and all $j \in I^g\backslash\{l\}$, we know from 1.(ii) that there is $\hat{t}_j\hat{u}_j \in S_j$ where $P_{jg}\hat{t}_j = P_{lg}t_l$ and $P_{jg}\hat{u}_j = P_{lg}u_l$. Additionally, because of 1.(iii) and $\hat{t}_j\hat{u}_j \in S_j$, there exists u_j where $t_j u_j \in S_j$ (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. Note that, from the way groups and these architectures are defined, $\mathcal{G}$ does not contain any cycle. For the fixed but arbitrary index $l \in I$, consider then, for all $g', g'' \in \mathcal{G}_l$, any components $j' \in I^{g'}\backslash\{l\}$ and $j'' \in I^{g''}\backslash\{l\}$. Take the strings $t_{j'}u_{j'}$ and $t_{j''}u_{j''}$ obtained from $t_l u_l$ as described in the previous paragraph. The absence of cycles implies that these strings do not have any events in common, so $P_{j'}^{-1}u_{j'} \cap P_{j''}^{-1}u_{j''} \neq \emptyset$. Moreover, we can apply the

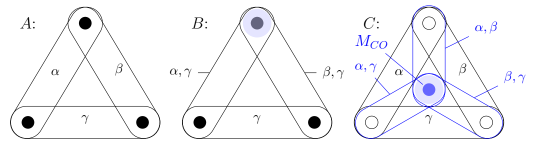

Fig. 3. Cyclic architectures without (A) and with lookout (B and C).

same arguments from the previous paragraph inductively on all $i \in I$ to obtain words u_i such that $t_i u_i \in S_i$ and $\bigcap_{i\in I} P_i^{-1} u_i \neq \emptyset$. Starting from the arbitrary index $l \in I$ and the string $t_l u_l$, we take all $j \in I \backslash \{l\}$ that are in the same groups as l and obtain $t_j u_j$. Then, we take all $p \in I \backslash \{j\}$ that are in the same groups as j, except for the groups considered in the previous inductive step. That is, for all $g \in \mathcal{G}_j \backslash \mathcal{G}_l$ and for all $p \in I^g \backslash \{j\}$, we can get strings $t_p u_p$ that also do not cause conflict among themselves and the previously obtained strings. We keep doing so till all $i \in I$ are considered. Note that in this induction we do revisit indexes due to the absence of cycles in these architectures. In other words, there is only one sequence of groups that can connect l to any other component $i \in I$ in the system. This guarantees the intersection $\bigcap_{i\in I} P_i^{-1} u_i$ to be nonempty, from where we can take a string u. 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 cycles, 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 l 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 c acts as a coordinator, as it observes all shared events in the plant. From component l, we can choose c as the first step of the induction (as c, being a lookout, shares events with l). Then, u_c serves as a template that imposes a feasible sequence between all shared events in the plant, guaranteeing that we can follow the induction such that $\bigcap_{i\in I} P_i^{-1} u_i \neq \emptyset$. ■

B. Coordinators for Non-Admissible Architectures.

Unfortunately, there are very simple architectures for which admissibility cannot be guaranteed. One example is defined below, and Fig. 4 illustrates a plant with such architecture where Hyp. 1 is not valid.

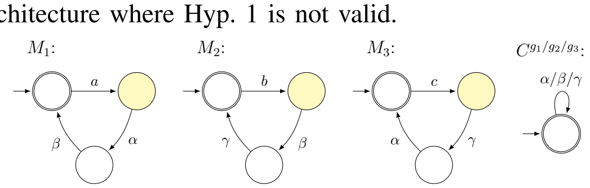

Fig. 4. Cyclic-architecture plant from Fig. 3 (A) requiring a coordinator.

Definition 9: A set of groups $\mathcal{G} = \{g_u\}_U$ forms a *cyclic-without-lookout architecture* if $\mathcal{G}$ is a minimal cycle of groups (as in Def. 7) but there is no lookout component, i.e., there is no $i \in I$ such that $\Sigma^{\mathcal{G}} = \bigcup_{g\in\mathcal{G}} \Sigma^g = \bigcup_{j\in I} \Sigma_j^{\mathrm{s}} \subseteq \Sigma_i$.

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 $\mathbf{M}_{\mathrm{co}}$ in the plant $\mathcal{M}$, as follows. Define $\mathbf{M}_{\mathrm{co}}$ 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 original minimal cycle, i.e., $\Sigma^{\mathcal{G}}$. Note that more minimal cycles are formed with the addition of the coordinator. In this new architecture (see Fig. 3 (C)) we can guarantee nonconflict, as per Prop. 1, and the supervisor $\mathbf{S}_{\mathrm{co}}$ is then a coordinator for the shared events in the cycle.

C. Connecting Admissible Architectures.

Definition 10: Let $\{\mathcal{M}_z\}_Z$ be a set of plants, where Z is a set of indexes. For each plant $\mathcal{M}_z$, denote by $\mathcal{G}^z$ the set of all its different groups. For different $z, z' \in Z$, consider the systems may have common components, i.e., $\mathcal{M}_z \cap \mathcal{M}_{z'} \neq \emptyset$. We then say the plant $\mathcal{M} = \bigcup_{z\in Z} \mathcal{M}_z$ has a *mixed architecture*.

Proposition 2: For a plant $\mathcal{M}$ with a mixed architecture, $\mathcal{M}$ is admissible, i.e., Hyp. 1 is valid, if

1) for each $z \in Z$, $\mathcal{M}_z$ is either a single-group, a linear, a star or a cyclic-with-lookout architecture, and
2) for all minimal cycle $\mathfrak{C} \subseteq \mathcal{G}$, there is $z \in Z$ such that $\mathfrak{C} \subseteq \mathcal{G}^z$. □

Proof: For this proof, we can use the same inductive 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. ■

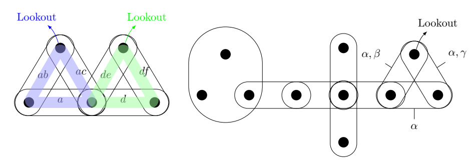

Fig. 5. Examples of admissible mixed architectures.

VIII. Decentralised Supervisory Control

The goal of CBSS is to compute automata $\mathbf{S}_i$ that allow the extraction of supervisors f_i to solve Problem 1. We first discuss in Sec. VIII-A how cooperative supervisors can be obtained 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 $\Sigma_{i,\mathrm{uc}}^{\mathrm{p}}$. Unfortunately, this does not imply that S_i is controllable with respect to $\Sigma_{i,\mathrm{cp}}^{\mathrm{s}}$. 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 $f_i : \mathcal{L}(\mathbf{M}_i) \to \Gamma_i$ which only disables locally controllable events in the set $\Sigma_{i,\mathrm{c}}$. That is, for all $s \in \mathcal{L}(\mathbf{M}_i)$, define $f_i(s) = \{\sigma \in \Sigma_i \mid s\sigma \in \overline{S_i}\} \cup \Sigma_{i,\mathrm{cp}}^{\mathrm{s}} \subseteq \Sigma_i$. 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 $\theta \in \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$; however, this is not expressed in the contract $\mathbf{C}^g_i$, 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 $d_{i\to} = d_{i\to}\big(w_i\big)$ is immediately updated to $d_{i\to} = d_{i\to}\big(w_i\alpha\big)$ 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 $\{\mathbf{S}^0_i\}_I$ be automata satisfying Cond. 1, and $(\{\mathbf{S}_i\}_I) = \textsc{Negotiation}(\{\mathbf{S}^0_i\}_I)$. 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 $f_i(w_i, d_{\to i}) = \{\sigma \,|\, w_i\sigma \in \overline{S_i} \text{ and } (\sigma \in \Sigma^{\mathrm{s}}_{i,\mathrm{c}} \to \mathrm{d}^\sigma \notin d_{\to i})\} \cup \Sigma^{\mathrm{s}}_{i,\mathrm{cp}}$.

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. 2, 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 $\mathbf{S}_i$ are nonempty automata (for all $i \in I$), 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) $\|_{i\in I}\, \mathcal{L}(f_i/\mathbf{M}_i) = \overline{\|_{i\in I}\, \mathcal{L}_m(f_i/\mathbf{M}_i)}$

(controllability taken w.r.t. $\mathcal{L}(\|_{i\in I}\, \mathbf{M}_i)$ 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 $\textsc{Negotiation}$. Secondly, each $\mathbf{S}_i$ outputted by CBSS satisfies Cond. 1, because: $\mathbf{K}_i$ satisfies Cond. 1, $\mathbf{S}_i$ outputted by $\textsc{Negotiation}$ satisfies it (see Lem. 1), and the output of $\textsc{EnforceRU}$ also satisfies it (Rem. 2). Thanks to $\textsc{EnforceRU}$ ([4]), for all $i \in I$ we have that $\mathbf{S}_i$ outputted by CBSS is unambiguous with respect to Σ^g_i, for all $g \in \mathcal{G}_i$ (Rem. 2). 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 $\|_{i\in I}\, S_i = \sup\mathcal{C}(\|_{i\in I}\, \tilde{S}_i)$, 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 $\sup\mathcal{C}(\|_{i\in I}\, \tilde{S}_i) \subseteq \sup\mathcal{C}(\|_{i\in I}\, K_i)$, 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.