Team Automata: Overview and Roadmap

Maurice H. ter Beek[1(✉)], Rolf Hennicker[2], and José Proença[3]

[1] CNR–ISTI, Pisa, Italy
maurice.terbeek@isti.cnr.it
[2] LMU Munich, Munich, Germany
hennicker@ifi.lmu.de
[3] CISTER, Faculty of Sciences, University of Porto, Porto, Portugal
jose.proenca@fc.up.pt

Abstract. Team Automata is a formalism for interacting component-based systems proposed in 1997, whereby multiple sending and receiving actions from concurrent automata can synchronise. During the past 25+ years, team automata have been studied and applied in many different contexts, involving 25+ researchers and resulting in 25+ publications. In this paper, we first revisit the specific notion of synchronisation and composition of team automata, relating it to other relevant *coordination models*, such as Reo, BIP, Contract Automata, Choreography Automata, and Multi-Party Session Types. We then identify several aspects that have recently been investigated for team automata and related models. These include *communication properties* (which are the properties of interest?), *realisability* (how to decompose a global model into local components?) and *tool support* (what has been automatised or implemented?). Our presentation of these aspects provides a snapshot of the most recent trends in research on team automata, and delineates a roadmap for future research, both for team automata and for related formalisms.

1 Introduction

Team automata (TA) were first proposed at the 1997 ACM SIGGROUP Conference on Supporting Group Work [82] for modelling components of groupware systems and their interconnections. They were inspired by Input/Output (I/O) automata [110] and in particular inherit their distinction between internal and external (i.e., input and output) actions used for communication with the environment (i.e., other I/O automata). Technically, team automata are an extension of I/O automata, since a number of the restrictions of I/O automata were dropped for more flexible modelling of several kinds of interactions in groupware systems. The underlying philosophy is that automata cooperate and collaborate by jointly executing (synchronising) transitions with the same action label (but possibly of different nature, i.e., input or output) as agreed upon upfront. They can be composed using a synchronous product construction that defines a unique composite automaton, the transitions of which are exactly those combinations of component transitions that represent a synchronisation on a common action by all the components that share that action. The effect of a synchronously executed action on the state of the composed automaton is described in terms of

Published by Springer Nature Switzerland AG 2024
I. Castellani and F. Tiezzi (Eds.): COORDINATION 2024, LNCS 14676, pp. 161–198, 2024.
https://doi.org/10.1007/978-3-031-62697-5_10

the local state changes of the automata that take part in the synchronisation. The automata not involved remain idle and their current states are unaffected.

Team automata were formally defined in Computer Supported Cooperative Work (CSCW)—The Journal of Collaborative Computing [41], in terms of component automata that synchronise on certain executions of actions. Unlike I/O automata, team automata impose hardly any restrictions on the role of actions in components and their composition is not limited to the synchronous product. Composing a team automaton requires defining its transitions by providing the actions and synchronisations that can take place from the combined states of the components. Each team automaton is thus a composite automaton defined over component automata. However, a given fixed set of component automata does not define a single unique team automaton, but rather a range of team automata, one for each choice of the team's transitions (individual or synchronising transitions from the component automata). This is in contrast with the usual synchronous product construction. The distinguishing feature of team automata is this very loose nature of synchronisation according to which specific synchronisation policies can be determined, defining how many component automata can participate in the synchronised execution of a shared external action, either as a sender (i.e., via an output action) or as a receiver (i.e., via an input action). This flexibility makes team automata capable of capturing in a precise manner a variety of notions related to coordination in distributed systems (of systems).

To illustrate this, consider the Race example in Fig. 1, borrowed from [31], which is meant to model a controller Ctrl that wants to *simultaneously* send to runners R1 and R2 a start message, after which it is able to receive from each runner *separately* a finish message once that runner *individually* has run. Here and in all subsequent examples and figures, components have exactly one initial state, indicated by a small incoming arrow head, and typically denoted by 0, and external actions may be prefixed by "!" (for output) or "?" (for input).

It is important to note that the synchronous product (as used in I/O automata and many other formalisms) of these three automata has a deadlock: after synchronisation of the three start transitions, Ctrl is blocked in state 1 until both R1 and R2 have executed their run action; at that point, full synchronisation of the finish transitions leads to a deadlock, with Ctrl in state 2 and R1 and R2 in their initial state. Team automata allow to exclude the latter synchronisation, yet at the same time enforcing the full synchronisation of start.

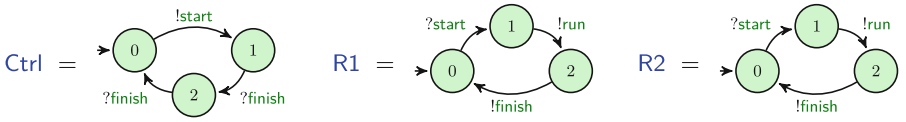


Fig. 1. Race example: a controller Ctrl and two runners R1 and R2

Contribution. We first revisit the specific notion of synchronisation and composition of team automata (Sect. 2). Next, we relate team automata to other coordination models frequently presented at the COORDINATION conferences,

such as Reo, BIP, Contract Automata, Choreography Automata, and Multi-Party Session Types (cf., e.g., [10,11,20,22,31,46]), inspired by a preliminary comparison in [119] (Sect. 3). We compare them by giving for each formalism (1) the definition, means of (2) composition (via synchronisation), (3) a model of the Race example, (4) a brief relation with team automata, and (5) tool support.

Table 1. Coordination formalisms and aspects analysed in this paper

Coordination Formalism (Sects. 2 & 3)	Communication Properties (Sect. 4)	Realisability (Sect. 5)	Verification (Sects. 4 & 5)	Supporting Tools (Sects. 3, 4 & 5)	Variability (Sect. 6)	Data (Sect. 6)
Team Automata [41,82]	✓	✓	✓	✓	✓	
Reo via Port Automata [2,102]			✓	✓	✓	✓
BIP [26,64]			✓	✓	✓	✓
Contract Automata [18,24]	✓	✓	✓	✓	✓	
Choreography Automata [10,12]	✓	✓	✓	✓		
Multi-Party Session Types [126,127]		✓	✓		✓	✓

We then focus on two aspects of team automata that we investigated during the last five years: *communication properties* (Sect. 4) and *realisability* (Sect. 5).

§4 We report results on compliance with communication requirements in terms of receptiveness (no message loss) and responsiveness (no indefinite waiting), give a thorough comparison with other compatibility notions, incl. deadlock-freedom, and give a roadmap for future work on communication properties.

§5 We report results on the decomposition (realisability) of a global interaction model in terms of a (possibly distributed) system of component automata coordinated according to a given synchronisation type specification. In particular, we provide a revised and extended comparison of our approach with that of Castellani et al. [73] and a roadmap for future work on realisability.

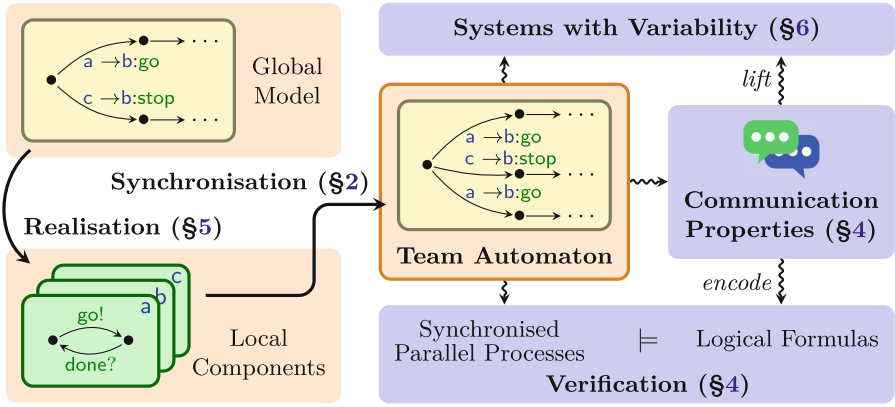


Fig. 2. Aspects of team automata addressed in this paper

Finally, we mention other aspects of team automata and of some of the related coordination models (Sect. 6) and conclude (Sect. 7). Table 1 shows the relations between the formalisms and aspects discussed in this paper. Figure 2 summarises this paper's contribution. Appendix A lists selected team automata publications.

2 Team Automata in a Nutshell

Team automata were originally introduced by Ellis [82] and formally defined in [41]. They form an automaton model for systems of reactive components that differentiate input (passive), output (active), and internal (privately active) actions. In this section, we recall the basic notions of (extended) team automata.

A *labelled transition system* (LTS) is a tuple $\mathcal{L} = (Q, q_0, \Sigma, E)$ such that Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite set of action labels, and $E \subseteq Q \times \Sigma \times Q$ is a transition relation. Given an LTS $\mathcal{L}$, we write $q \xrightarrow{a}_{\mathcal{L}} q'$, or shortly $q \xrightarrow{a} q'$, to denote $(q, a, q') \in E$. Similarly, we write $q \xrightarrow{a}_{\mathcal{L}}$ to denote that a is *enabled* in $\mathcal{L}$ at state q, i.e., there exists $q' \in Q$ such that $q \xrightarrow{a} q'$. For $\Gamma \subseteq \Sigma$, we write $q \xrightarrow{\Gamma}{}^* q'$ if there exist $q \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q'$ for some $n \geq 0$ and $a_1, \ldots, a_n \in \Gamma$. A state $q \in Q$ is *reachable by* Γ if $q_0 \xrightarrow{\Gamma}{}^* q$, it is *reachable* if $q_0 \xrightarrow{\Sigma}{}^* q$. The set of reachable states of $\mathcal{L}$ is denoted by $\mathcal{R}(\mathcal{L})$.

CA. A *component automaton* (CA) is an LTS $\mathcal{A} = (Q, q_0, \Sigma, E)$ such that $\Sigma = \Sigma^? \uplus \Sigma^! \uplus \Sigma^\tau$ is a set of *action labels* with disjoint sets $\Sigma^?$ of *input actions*, $\Sigma^!$ of *output actions*, and Σ^τ of *internal actions*. Cf. Fig. 1 for examples of CA.

Systems. A *system* is a pair $\mathcal{S} = (\mathcal{N}, (\mathcal{A}_n)_{n \in \mathcal{N}})$, with $\mathcal{N}$ a finite, nonempty set of names and $(\mathcal{A}_n)_{n \in \mathcal{N}}$ an $\mathcal{N}$-indexed family of CA $\mathcal{A}_n = (Q_n, q_{0,n}, \Sigma_n, E_n)$. Any system $\mathcal{S}$ induces an LTS defined by $\mathbf{lts}(\mathcal{S}) = (Q, q_0, \Lambda(\mathcal{S}), E(\mathcal{S}))$, where $Q = \prod_{n \in \mathcal{N}} Q_n$ is the set of *system states*, $q_0 = (q_{0,n})_{n \in \mathcal{N}}$ is the *initial system state*, $\Lambda(\mathcal{S})$ is the set of *system labels*, and $E(\mathcal{S})$ is the set of *system transitions*. Each system state $q \in Q$ is an $\mathcal{N}$-indexed family $(q_n)_{n \in \mathcal{N}}$ of local CA states $q_n \in Q_n$. The definitions of $\Lambda(\mathcal{S})$ and $E(\mathcal{S})$ follow after that of *system action*.

System Actions. The set of *system actions* $\Sigma = \bigcup_{n \in \mathcal{N}} \Sigma_n$ determines actions that will be part of system labels. Within Σ we identify $\Sigma^\bullet = \bigcup_{n \in \mathcal{N}} \Sigma_n^? \cap \bigcup_{n \in \mathcal{N}} \Sigma_n^!$ as the set of *communicating actions*. Hence, an action $a \in \Sigma$ is communicating if it occurs in (at least) one set Σ_k of action labels as an input action and in (at least) one set Σ_ℓ of action labels as an output action. The system is *closed* if all non-communicating actions are internal component actions. For ease of presentation, we assume in this paper that systems are closed.

System Labels. We use *system labels* to indicate which components participate (simultaneously) in the execution of a system action. There are two kinds of system labels. In a system label of the form $(\mathsf{out}, a, \mathsf{in})$, out represents the set of senders of *outputs* and in the set of receivers of *inputs* that synchronise on the action $a \in \Sigma^\bullet$. Either out or in can be empty, but not both. A system label of

the form (n, a) indicates that component n executes an internal action $a \in \Sigma_n^\tau$. Formally, the set $\Lambda(\mathcal{S})$ of system labels of $\mathcal{S}$ is defined as follows:

$$\Lambda(\mathcal{S}) = \{\, (\mathsf{out}, a, \mathsf{in}) \mid \varnothing \neq (\mathsf{out} \cup \mathsf{in}) \subseteq \mathcal{N},\ \forall_{n \in \mathsf{out}} \cdot a \in \Sigma_n^!,\ \forall_{n \in \mathsf{in}} \cdot a \in \Sigma_n^? \,\}$$
$$\cup\, \{\, (n, a) \mid n \in \mathcal{N},\ a \in \Sigma_n^\tau \,\}$$

Note that $\Lambda(\mathcal{S})$ depends only on $\mathcal{N}$ and on the sets Σ_n of action labels for each $n \in \mathcal{N}$. If $\mathsf{out} = \{n\}$ is a singleton, we write (n, a, in) instead of $(\{n\}, a, \mathsf{in})$, and similarly for singleton sets in. In all figures and examples, interactions $(\mathsf{out}, a, \mathsf{in})$ are presented by the notation $\mathsf{out} \rightarrow \mathsf{in} : \mathsf{a}$ and internal labels (n, a) by $\mathsf{n} : \mathsf{a}$. System labels provide an appropriate means to describe which components in a system execute, possibly together, a computation step, i.e., a system transition.

System Transitions. A *system transition* $t \in E(\mathcal{S})$ has the form $(q_n)_{n \in \mathcal{N}} \xrightarrow{\lambda}_{\mathsf{lts}(\mathcal{S})} (q'_n)_{n \in \mathcal{N}}$ such that $\lambda \in \Lambda(\mathcal{S})$ and

- either $\lambda = (\mathsf{out}, a, \mathsf{in})$ and:
  - $q_n \xrightarrow{a}_{\mathcal{A}_n} q'_n$, for all $n \in \mathsf{out} \cup \mathsf{in}$, and $q_m = q'_m$, for all $m \in \mathcal{N} \backslash (\mathsf{out} \cup \mathsf{in})$;
- or $\lambda = (n, a)$, $a \in \Sigma_n^\tau$ is an internal action of some component $n \in \mathcal{N}$, and:
  - $q_n \xrightarrow{a}_{\mathcal{A}_n} q'_n$ and $q_m = q'_m$, for all $m \in \mathcal{N} \backslash \{n\}$.

We write Λ and E instead of $\Lambda(\mathcal{S})$ and $E(\mathcal{S})$, respectively, if $\mathcal{S}$ is clear from the context. Surely, at most the components that are in a local state where action a is locally enabled can participate in a system transition for a. Since, by definition of system labels, $(\mathsf{out} \cup \mathsf{in}) \neq \varnothing$, at least one component participates in any system transition. Given a system transition $t = q \xrightarrow{\lambda}_{\mathsf{lts}(\mathcal{S})} q'$, we write $t.\lambda$ for λ.

Example 1. The Race system in Fig. 1 has both, desired system transitions such as $(0,0,0) \xrightarrow{(\mathsf{Ctrl},\mathsf{start},\{\mathsf{R1},\mathsf{R2}\})} (1,1,1)$ and $(1,2,2) \xrightarrow{(\mathsf{R1},\mathsf{finish},\mathsf{Ctrl})} (2,0,2)$, and undesired ones like $(0,0,0) \xrightarrow{(\mathsf{Ctrl},\mathsf{start},\varnothing)} (1,0,0)$, and $(1,2,2) \xrightarrow{(\{\mathsf{R1},\mathsf{R2}\},\mathsf{finish},\mathsf{Ctrl})} (2,0,0)$. The LTS of the Race system, denoted by $\mathsf{lts}(\mathsf{Race})$, contains all possible system transitions. As mentioned in the Introduction, the latter two are undesired since the controller is supposed to start both runners simultaneously, whereas they should finish individually. These and other system transitions will be discarded based on synchronisation restrictions for teams considered next. ▷

Team Automata. Synchronisation types specify which synchronisations of components are admissible in a specific system $\mathcal{S}$. A *synchronisation type* $(O, I) \in \mathsf{Intv} \times \mathsf{Intv}$ is a pair of intervals O and I which determine the number of outputs and inputs that can participate in a communication. Each interval has the form $[min, max]$ with $min \in \mathbb{N}$ and $max \in \mathbb{N} \cup \{*\}$ where $*$ denotes 0 or more participants. We write $x \in [min, max]$ if $min \leq x \leq max$ and $x \in [min, *]$ if $x \geq min$.

A *synchronisation type specification* (STS) over $\mathcal{S}$ is a function $\mathbf{st} : \Sigma^\bullet \rightarrow \mathsf{Intv} \times \mathsf{Intv}$ that assigns to any communicating action a an individual synchronisation type $\mathbf{st}(a)$. A system label $\lambda = (\mathsf{out}, a, \mathsf{in})$ *satisfies* $\mathbf{st}(a) = (O, I)$, denoted

by $\lambda \models \mathbf{st}(a)$, if $|\mathsf{out}| \in O \wedge |\mathsf{in}| \in I$. Each STS $\mathbf{st}$ generates the following subsets $\Lambda(\mathcal{S}, \mathbf{st})$ of system labels and $E(\mathcal{S}, \mathbf{st})$ of corresponding system transitions.

$$\Lambda(\mathcal{S}, \mathbf{st}) = \{\, \lambda \in \Lambda \mid \lambda = (\mathsf{out}, a, \mathsf{in}) \Rightarrow \lambda \models \mathbf{st}(a) \,\}$$
$$E(\mathcal{S}, \mathbf{st}) = \{\, t \in E \mid t.\lambda \in \Lambda(\mathcal{S}, \mathbf{st}) \,\}$$

Thus, for communicating actions, the set of system transitions is restricted to those transitions whose labels respect the synchronisation type of their communicating action. For internal actions no restriction is applied, since an internal action of a component can always be executed when it is locally enabled.

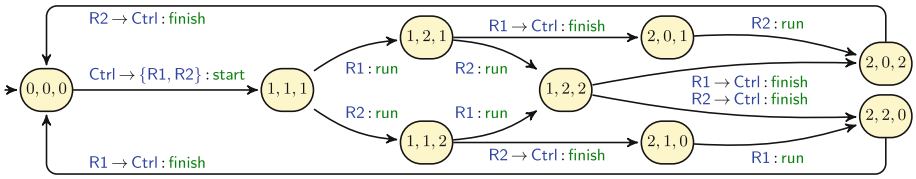


Fig. 3. Team automaton of the Race system example in Fig. 1

Components interacting in accordance with an STS $\mathbf{st}$ over a system $\mathcal{S}$ are seen as a team whose behaviour is represented by the (extended) *team automaton* (TA) $\mathbf{ta}(\mathcal{S}, \mathbf{st})$ generated over $\mathcal{S}$ by $\mathbf{st}$ and defined by the LTS

$$\mathbf{ta}(\mathcal{S}, \mathbf{st}) = (Q, q_0, \Lambda(\mathcal{S}, \mathbf{st}), E(\mathcal{S}, \mathbf{st})).^{1}$$

We write $\Lambda(\mathcal{S}, \mathbf{st})$ and $E(\mathcal{S}, \mathbf{st})$ instead of $\Lambda(\mathcal{S}, \mathbf{st})$ and $E(\mathcal{S}, \mathbf{st})$, respectively, if $\mathcal{S}$ is clear from the context, and assume that $\Lambda(\mathcal{S}, \mathbf{st}) \neq \varnothing$. Labels in $\Lambda(\mathcal{S}, \mathbf{st})$ are called *team labels* and transitions in $E(\mathcal{S}, \mathbf{st})$ are called *team transitions*.

Example 2. For the Race system $\mathcal{S}_{\mathsf{Race}}$ in Fig. 1. we define the runners to *start* simultaneously and *finish* individually by the STS $\mathbf{st}_{\mathsf{Race}} = \{\mathsf{start} \mapsto ([1,1],[2,2]), \mathsf{finish} \mapsto ([1,1],[1,1])\}$. The resulting TA $\mathbf{ta}(\mathbf{st}_{\mathsf{Race}}, \mathcal{S}_{\mathsf{Race}})$ is shown in Fig. 3, with interactions (n, a, m) written as $\mathsf{n} \to \mathsf{m} : \mathsf{a}$ and internal labels (n, a) as $\mathsf{n} : \mathsf{a}$. ▷

3 Related Coordination Formalisms

In this section, we introduce a selection of formal coordination models and languages and compare them to team automata by providing, for each formalism, (1) the definition of the variant considered here, (2) the definition of composition (via synchronisation), (3) a possible model of our Race example in the formalism, (4) a brief relation with team automata, and (5) existing tool support.

[1] Starting with [44], we use the system labels $(\mathsf{out}, a, \mathsf{in})$ in $\Lambda(\mathcal{S}, \mathbf{st})$ as the actions in team transitions of what we coined extended team automata (ETA). This is the main difference with the 'classical' team automata from [41,82] and subsequent papers, where actions $a \in \Sigma$ have been used in team transitions. However, to study communication properties [36], compositionality [44] and realisability [47], explicit rendering of the CA that actually participate in a transition of the team turned out useful.

3.1 Reo via Port Automata

Reo [2,96] is a coordination language to specify and compose *connectors*, i.e., patterns of valid synchronous interactions of ports of components or other connectors. For example, a FIFO1 connector has two ports: a source port to receive data and a sink port to send data. It initially allows the source port to interact while blocking the sink port, after which it allows the sink port to interact while blocking the source port. The duplicator connector has a single source port and two sink ports, only allowing all ports to interact at the same time or none.

Constraint automata [7] is a reference model for Reo's semantics [96]. We use a simplified variant called *port automata* [102], which abstracts away from data constraints, focusing on *synchronisation* and *composition*.

Definition. A port automaton (PA) $P = (Q, \Sigma, \rightarrow, Q_0)$ consists of a set of states Q, a set of ports Σ, a transition relation $\rightarrow \subseteq Q \times 2^{\Sigma} \times Q$, and a set of initial states $Q_0 \subseteq Q$. We have $(q, \{a, b\}, q') \in \rightarrow$ when the PA can evolve from state q to q' by *simultaneously executing* ports a and b.

Composition. Two PA $(Q_1, \Sigma_1, \rightarrow_1, Q_{0,1})$ and $(Q_2, \Sigma_2, \rightarrow_2, Q_{0,2})$ with shared ports and disjoint states can be composed by forcing the shared ports to synchronise. Then the composition yields a new PA $(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \rightarrow, Q_{0,1} \times Q_{0,2})$ where $\rightarrow$ is defined by the following rules (and the symmetric of the second rule):

$$\frac{q_1 \xrightarrow{\sigma_1}_1 q_1' \quad q_2 \xrightarrow{\sigma_2}_2 q_2' \quad \sigma_1 \cap \Sigma_2 = \sigma_2 \cap \Sigma_1}{\langle q_1, q_2 \rangle \xrightarrow{\sigma_1 \cup \sigma_2} \langle q_1', q_2' \rangle} \qquad \frac{q_1 \xrightarrow{\sigma_1}_1 q_1' \quad \sigma_1 \cap \Sigma_2 = \varnothing}{\langle q_1, q_2 \rangle \xrightarrow{\sigma_1} \langle q_1', q_2 \rangle}$$

Note that the condition $\sigma_1 \cap \Sigma_2 = \sigma_2 \cap \Sigma_1$ intuitively means that any shared action of σ_1 must be available in σ_2 as well, and vice versa.

Race in Port Automata. Unlike TA, Reo's focus is on building connectors by composing simpler connectors, instead of composing components, to produce a system. Hence one could produce a Reo connector by composing a set of simpler primitive connectors that, once composed, would allow only the valid interaction patterns of our Race example. A possible connector is depicted in Fig. 4, borrowed from [119], which composes two FIFO1 connectors (), two synchronous barriers (), three replicators ( after $\mathsf{start}_{\mathsf{Ctrl}}$, $\mathsf{finish}_{\mathsf{R1}}$, and $\mathsf{finish}_{\mathsf{R2}}$), and one interleaving merger ( before $\mathsf{finish}_{\mathsf{Ctrl}}$). Each has a PA for its semantics, and their composition yields the PA depicted on the right of Fig. 4.

Fig. 4. Reo connector for the Race example (left) and its semantics as a PA (right), after hiding internal ports shared among sub-connectors

Brief Relation with TA. Many variants of constraint automata exist [96, Sect. 3.2.2], some distinguishing inputs from outputs as in TA. Synchronisation types in TA restrict the number of inputs and outputs of ports with shared names; synchronisation in PA force how. No variant uses a similar notion to TA's synchronisation types, although they can be expressed using intermediate ports.

Tool Support. There are tools to *analyse*, *edit*, *visualise*, and *execute* Reo connectors. Analyses include model checking, using either the dedicated model checker Vereofy [101] or encoding Reo into mCRL2 [103,118], and simulation of extensions with parameters [120] and with reactive programming notions [122], many accessible online at http://arcatools.org/reo. Editors and visualisation engines include an Eclipse-based implementation [4] and editors based on JavaScript that run in a browser [76,130]. Execution engines for Reo include a Java-based implementation [78] and a distributed engine using actors in Scala [121,123].

3.2 BIP Without Priorities

BIP [26,64] is formal language to specify architectures for interacting components. A program describes the **B**ehaviour of each component, the valid **I**nteractions between their ports, and the **P**riority among interactions. Multiple formal models for specifying interactions exist, such as an algebra of connectors [64]. We follow the formalisation of the operational semantics of Bliudze and Sifakis [64], disregarding the priority aspect for simplicity. In this paper, ports of BIP are called actions to facilitate the comparison with TA.

Definition. A *local behaviour* B is given by an LTS (Q, q_0, Σ, E), with set Q of states, initial state q_0, labels Σ for actions, and transition relation $E : Q \times \Sigma \times Q$.

Composition into a BIP Program. A BIP program without priorities is a pair consisting of (1) an $\mathcal{N}$-indexed family of local behaviours $(B_n)_{n\in\mathcal{N}} = (Q_n, q_{0,n}, \Sigma_n, E_n)$, with pairwise disjoint action sets Σ_n, one for each agent n, and (2) a set of valid interactions I where each interaction $a \in I$ is a family $(a_n)_{n\in N}$ such that $N \subseteq \mathcal{N}$ and $a_n \in \Sigma_n$, for all $n \in N$, and I is a set of such interactions. The semantics of a BIP program BP is given by an LTS $\mathbf{lts}(BP) = (\prod_{n\in\mathcal{N}} Q_n, (q_{0,n})_{n\in\mathcal{N}}, I, E)$, where E is defined by the following rule:

$$\frac{a = (a_n)_{n\in N} \in I \ \wedge \ \forall n \in N : \left(q_n \xrightarrow{a_n}_{B_n} q'_n\right) \ \wedge \ \forall n \in \mathcal{N} \setminus N : q_n = q'_n}{(q_n)_{n\in\mathcal{N}} \xrightarrow{a}_{\mathbf{lts}(BP)} (q'_n)_{n\in\mathcal{N}}}$$

Race in BIP. The encoding of our Race example in BIP without priorities is depicted in Fig. 5. It consists of the three components on the left, where we use a set notation for each interaction in I. This set I of valid interactions generalises the synchronisation policies of TA, imposing all start actions to synchronise and the finish actions to synchronise two at a time between the controller and one runner. The LTS of the BIP program is the same as the PA on the right side of Fig. 4, omitting the internal action run included in the TA model of the Race example.

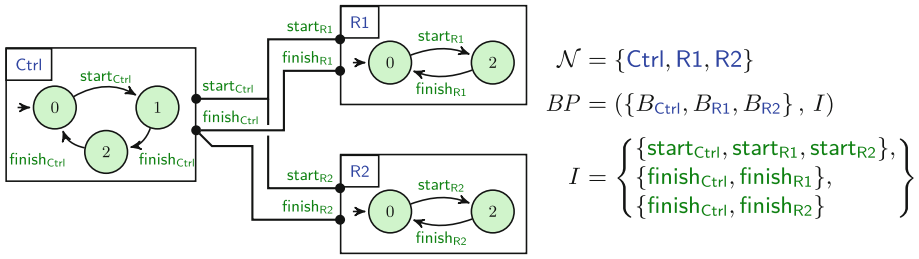


Fig. 5. Race example in BIP: individual components are labelled by actions, restricted to the interactions allowed by I, imposed by the (stateless) connector

Brief Relation with TA. We have previously [31] compared TA with BIP [26], describing how some explicit patterns of interaction of BIP, such as broadcasts, are modelled in TA. In this paper, we used precise formalisations of TA and BIP without priorities [104], presented in a similar style to facilitate the comparison.

There are some technical core differences between these formalisations of BIP and TA. BIP's formalisation does not include explicit internal actions, although they do exist at implementation level (e.g., in JavaBIP [63]). BIP's synchronisation mechanism ignores inputs and outputs. However, the flow of data at each interaction is sometimes described orthogonally [63], where ports can either be *enforceable* by the environment (similar to input actions) or *spontaneous* by the components (similar to output actions). A more thorough comparison of the expressiveness of different BIP formalisations is given by Baranov and Bliudze [8]. These differences reflect a different focus: less emphasis on communication properties (Sect. 4), internal behaviour of local components, and realisability notions (Sect. 5); yet more focus on the exploration of different formalisms to compose interactions and programs, supported by tools to connect to running systems [63].

Similar to TA's synchronisation types, BIP has a formalisation parameterised on the number of components [111]. Ports can have multiple instances, and are enriched with a bound on the number of allowed agents they can synchronise with, and a bound on the number of interactions they can be involved in.

Tool Support. Several tools exist for BIP, including verification tools that traverse the state space of BIP programs [62] or use the VerCors model checker [61],

and a toolset LALT-BIP for verifying freedom from global and local deadlocks [6]. BIP also has a C++ reference engine [27] and a Java engine called JavaBIP [63].

3.3 Contract Automata

Contract automata [18,24] are a finite state automata dialect proposed to model multi-party composition of contracts that can perform *request* or *offer* actions, which need to match to achieve *agreement* among a composition of contracts. Contract automata have been equipped with variability in [15,19] by *modalities* to specify when an action *must* be matched (necessary) and when it *may* be withdrawn (optional) in a composition, and with *real-time constraints* in [17]. We first give a definition without modalities, silent actions or variability constraints.

Definition. Let $\overline{v} = [e_1, \ldots, e_n]$ be a vector of rank $n \geq 1$ and let $v(i)$ denote its ith element. A *contract automaton* of rank $n \geq 1$ is a tuple $(Q, \overline{q}_0, \Sigma^r, \Sigma^o, \rightarrow, F)$ such that $Q = Q_1 \times \ldots \times Q_n$ is the product of finite sets of states, $\overline{q}_0 \in Q$ is the initial state, Σ^r is a finite set of requests, Σ^o is a finite set of offers, $\rightarrow \subseteq Q \times (\Sigma^r \cup \Sigma^o \cup \{-\})^n \times Q$ is a set of transitions constrained as follows next, and $F \subseteq Q$ a the set of final states. A transition $(\overline{q}, \overline{a}, \overline{q}') \in \rightarrow$ is such that $\overline{a}$ is either a single offer (i.e., $\exists i \,.\, \overline{a}(i) = !a \in \Sigma^o$ and $\forall j \neq i \,.\, \overline{a}(j) = -$), a single request (i.e., $\exists i \,.\, \overline{a}(i) = ?a \in \Sigma^r$ and $\forall j \neq i \,.\, \overline{a}(j) = -$) or a single pair of matching request and offer (i.e., $\exists i, j \,.\, \overline{a}(i) = !a \wedge \overline{a}(j) = ?a$ and $\forall k \neq i, j \,.\, \overline{a}(k) = -$), and $\forall i \,.\, \overline{a}(i) = - \implies \overline{q}(i) = \overline{q}'(i)$.

Next we define contract automata with committed states as introduced in [21]: whenever a state $\overline{q}$ has a committed element $\overline{q}(i)$, then all outgoing transitions of $\overline{q}$ have a label $\overline{a}$ such that $\overline{a}(i) \neq -$, i.e., whenever the intermediate state of two concatenated transitions is committed, the two transitions are executed atomically: after the first transition has been executed, the second transition is executed prior to any other transition of any other service in the composition.

Composition. Composition of contracts is rendered through the composition of their contract automaton models by means of the *composition operator* $\otimes$, a variant of the synchronous product, which interleaves or matches the transitions of the component (contract) automata such that whenever two components are enabled to execute their respective request/offer action, then the match must happen. A composition is in *agreement* if each request is matched with an offer.

Race in Contract Automata. We use contract automata with committed states to mimick multi-party synchronisation (cf. Fig. 6). In their composition in Fig. 7, states $[\mathrm{C}, 1, 0]$ and $[\mathrm{C}, 0, 1]$ have a committed state, meaning that only outgoing transitions in which Ctrl changes state are permitted. In [21], silent (τ) actions are introduced, but we choose to model internal run actions as offer actions !run (which do not interfere with agreement) rather than silent actions τ_{run}.

Brief Relation with TA. In [25], contract automata are compared with communicating machines [66]. To guarantee that a composition corresponds to a

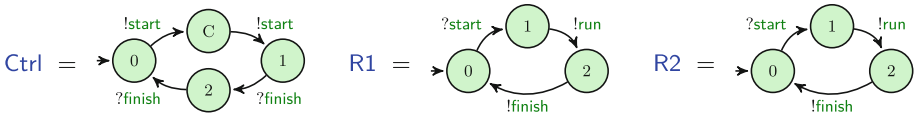


Fig. 6. Contract automata with committed states for the Race example

well-behaving (i.e., *realisable*) choreography, a *branching condition* is used. This condition requires contract automata to perform their offers independently of the other component automata in the composition. As noted in [15], this condition is related to the phenomenon of *state sharing* in team automata [83], meaning that system components influence potential synchronisations through their local (component) states even if not involved in the actual global (system) transition. While a synchronous product of (I/O) automata can directly be seen as a Petri net, for team automata this only holds for non-state-sharing vector team automata [52,68]. The relation between the branching condition of contract automata and (non-)state-sharing in (vector) team automata needs further study.

Tool Support. Contract automata are supported by a software API called Contract Automata Library (CATLib) [14], which a developer can exploit to specify contract automata and perform operations like composition and synthesis. The synthesis operation uses supervisory control theory [124], properly revisited in [23] for synthesising orchestrations and choreographies of contract automata. An application developed with CATLib is thus formally validated *by-construction* against well-behaving properties from the theory of contract automata [15,18,23]. CATLib was designed to be easily extendable to support related formalisms; it currently supports synchronous communicating machines [105].

3.4 Choreography Automata

Choreography Automata (ChorAut) [10] are automata with labels that describe interactions, including a sender, a receiver, and a message name. This section is less detailed than the previous ones due to the similarity with contract automata.

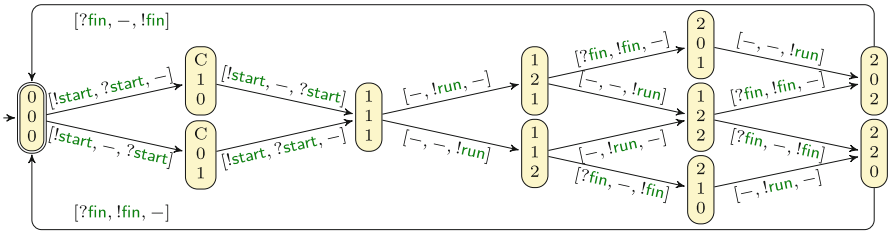


Fig. 7. Composition Ctrl ⊗ R1 ⊗ R2 of contract automata

Race in ChorAut. A ChorAut model of our Race example without the internal run actions is depicted in Fig. 8.

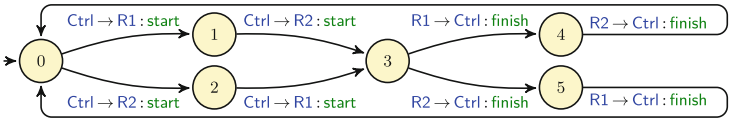


Fig. 8. ChorAut of the Race example (without internal run actions)

Brief Relation with TA. Internal actions are not captured by ChorAut (as in Reo and BIP's formalisations) and only binary synchronisations are supported (as in contract automata): each interaction has a single sender (the agent that *offers*) and a single receiver (the agent that *requests*). Desirable properties of ChorAut include deadlock-freedom, among others, focused on the language accepted by these automata [11]. Consequently, properties that rely on observational equivalence notions such as bisimilarity are not covered by these analyses.

Tool Support. Corinne [117] can be used to visualise ChorAut and to automatise operations like projection, composition, and checking for well-formedness.

3.5 Synchronous Multi-Party Session Types

Multi-Party Session Types (MPST) [126] are a family of formalisms based on calculi to describe communication protocols between multiple agents (*multiparty*). A *session* in these calculi represents a communication channel shared by a group of agents, to which they can read or write data. In MPST, each agent has a *behavioural type*, which describes the allowed patterns of reading-from and writing-to sessions, providing some compile-time guarantees for the concrete agents to follow the communication patterns. Most approaches distinguish (1) a *global type*, often a starting point, describing the composed system; and (2) the *local types*, often derived from the global type, describing the local view of each agent.

This paper uses the definitions of a simple synchronous MPST (SyncMPST) used by Seviri and Dezani-Ciancaglini [127] that only supports binary synchronisations. Other SyncMPST exist, some supporting multiple receivers [58] or multiple senders [95]. We opt to use a simpler model that can be compactly described and provides enough insights to relate SyncMPST with TA.

Definition. The syntax of a global type $\mathcal{G}$, a local type $\mathcal{L}$, and a system $\mathcal{S}$ are given by the grammars below. $\mathcal{G}$ and $\mathcal{L}$ definitions are interpreted coinductively, i.e., their solutions are both minimal and maximal fixpoints.

$$\mathcal{G} ::= \mathtt{end} \mid \mathtt{p} \to \mathtt{q} : \Gamma \qquad \mathcal{L} ::= 0 \mid \mathtt{p}!\Lambda \mid \mathtt{p}?\Lambda \qquad \mathcal{S} ::= \mathtt{p} \triangleright \mathcal{L}$$
$$\Gamma ::= \{\mathtt{a}_i.\mathcal{G}_i\}_{1 \le i \le k} \qquad \Lambda ::= \{\mathtt{a}_i.\mathcal{L}_i\}_{1 \le i \le k} \qquad \mid \mathcal{S} \parallel \mathcal{S}$$

The following conditions must hold: (1) branches Γ and Λ must have disjoint initial messages $\mathtt{a}_i$; (2) agents $\mathtt{p}$ in $\mathcal{S}$ appearing in the left of $\mathtt{p} \triangleright \mathcal{L}$ must be different and each $\mathcal{L}$ must not include $\mathtt{p}$; and (3) for every $\mathtt{p} \to \mathtt{q} : \Gamma$ and $\mathtt{r} \notin \{\mathtt{p}, \mathtt{q}\}$,

r *should not distinguish* any choice in Γ. The third point captures *projectability* [127], which is not formalised here and is closely related to realisability (cf. Sect. 5). Whenever clear we omit curly brackets and trailing `end` and `0`.

The semantics of a global type is given by the rules below. The semantics of a system is given by the composition of local types presented in the next paragraph. The system obtained by projecting a global type is guaranteed to accept the same language as the global type [127].

$$\mathsf{p} \to \mathsf{q} : \{\mathsf{a}.\mathcal{G}, \dots\} \xrightarrow{\mathsf{p}\to\mathsf{q}:\mathsf{a}} \mathcal{G} \qquad \frac{\{\mathsf{p},\mathsf{q}\} \cap \{\mathsf{r},\mathsf{s}\} = \varnothing \qquad \forall_{1\le i\le k} : \mathcal{G}_i \xrightarrow{\mathsf{p}\to\mathsf{q}:\mathsf{b}} \mathcal{G}'_i}{\mathsf{r} \to \mathsf{s} : \{\mathsf{a}_i.\mathcal{G}_i\}_{1\le i\le k} \xrightarrow{\mathsf{p}\to\mathsf{q}:\mathsf{b}} \mathsf{r} \to \mathsf{s} : \{\mathsf{a}_i.\mathcal{G}'_i\}_{1\le i\le k}}$$

Composition. A system $\mathcal{S}$ of agents, each with a given local type, evolves according to the following rule:

$$\mathsf{p} \rhd \mathsf{q}!\{\mathsf{a}.\mathcal{L}_\mathsf{p}, \dots\} \parallel \mathsf{q} \rhd \mathsf{p}?\{\mathsf{a}.\mathcal{L}_\mathsf{q}, \dots\} \parallel \mathcal{S} \xrightarrow{\mathsf{p}\to\mathsf{q}:\mathsf{a}} \mathsf{p} \rhd \mathcal{L}_\mathsf{p} \parallel \mathsf{q} \rhd \mathcal{L}_\mathsf{q} \parallel \mathcal{S}$$

Race in SyncMPST. The Race example cannot be directly modelled using this SyncMPST. We model a variant in Fig. 9, using only binary synchronisation and using distinct start_1 and start_2 to differentiate the choice in the branch. Some other SyncMPST approaches also consider non-binary synchronisations [58,95], which could support the synchronisation of start with all three participants. There is also a need to prefix every choice with a concrete message between participants, leading to the need for distinguishing start_1 from start_2. This requirement is common among most MPST, which lead to our variant of the Race where an early choice is taken with start_1 or start_2 about which runner finishes first. For simplicity our variation assumes that R1 receives the start earlier. Alternatively we could have allowed either runners to start, as done with contract automata (Fig. 6) and choreography automata (Fig. 8). This would lead to a duplication of the code (one for each option) and to the introduction of a new initial action that prefixes the choice of which runner starts.

$$\mathcal{G} = \mathsf{Ctrl} \to \mathsf{R1} : \mathsf{start}.\mathsf{Ctrl} \to \mathsf{R2} : \left\{ \begin{array}{l} \mathsf{start}_1.(\mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}.\mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish}.\mathcal{G}), \\ \mathsf{start}_2.(\mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish}.\mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}.\mathcal{G}) \end{array} \right\}$$

$$\mathcal{S} = \mathsf{Ctrl} \rhd \mathcal{L}_{\mathsf{Ctrl}} \parallel \mathsf{R1} \rhd \mathcal{L}_{\mathsf{R1}} \parallel \mathsf{R2} \rhd \mathcal{L}_{\mathsf{R2}}$$

$$L_{\mathsf{Ctrl}} = \mathsf{R1}!\mathsf{start}.\mathsf{R2}!\left\{\mathsf{start}_1.(\mathsf{R1}?\mathsf{finish}.\mathsf{R2}?\mathsf{finish}.\mathcal{L}_{\mathsf{Ctrl}})\,,\ \mathsf{start}_2.(\mathsf{R2}?\mathsf{finish}.\mathsf{R1}?\mathsf{finish}.\mathcal{L}_{\mathsf{Ctrl}})\right\}$$

$$\mathcal{L}_{\mathsf{R1}} = \mathsf{Ctrl}?\mathsf{start}.\mathsf{Ctrl}!\mathsf{finish}.\mathcal{L}_{\mathsf{R1}}$$

$$\mathcal{L}_{\mathsf{R2}} = \mathsf{Ctrl}?\left\{\mathsf{start}_1.\mathsf{Ctrl}!\mathsf{finish}.\mathcal{L}_{\mathsf{R2}}\,,\ \mathsf{start}_2.\mathsf{Ctrl}!\mathsf{finish}.\mathcal{L}_{\mathsf{R2}}\right\}$$

Fig. 9. Global type ($\mathcal{G}$) and system ($\mathcal{S}$) of a Race example variant in SyncMPST

Brief Relation with TA. SyncMPSTs support a relatively strict subset of interaction patterns. Consequently, many useful properties may be syntactically

verified, like deadlock-freedom or the preservation of behaviour after projection, at the cost of expressivity.

Regarding internal behaviour, neither global nor local types support internal actions. As an alternative to internal actions in types, many MPST variations describe a separate syntax for processes with data and control structures, and define well-typedness w.r.t. local types (cf., e.g., Bejleri and Yoshida [58]).

A rich variety of tools are built over variants of MPST. A recent survey by Yoshida [134] of MPST implementations over different programming languages reflects the focus on producing trustworthy distributed implementations.

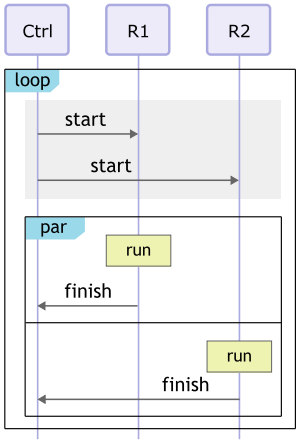
3.6 Other Coordination Formalisms

Many other coordination formalisms involve similar notions of composition and synchronisation of agent behaviour. These include the specification languages provided by model checkers such as Uppaal [57,107] and mCRL2 [5,85], as well as more generic formalisms like Petri nets [60,125], message sequence charts [89, 93], event structures [115,133] and I/O automata [99,110]. Without pretending completeness, we discuss some of these in this section.

Uppaal accepts systems modelled by (stochastic, timed) automata, with matching input-output actions that must synchronise (either 1-to-1 or 1-to-many). The latter requires a sender to synchronise with all, possibly zero, available receivers at that time, which differs from the synchronisation policies of TA. Contract automata's committed states stem from Uppaal's concept of committed states. Uppaal also provides partial support for priorities over actions, but not over interactions as in BIP.

mCRL2 accepts systems modelled as a parallel composition of algebraic processes, with special operators to allow the synchronous execution of groups of actions, and the restriction of given actions. This is powerful enough to enumerate all valid synchronisations between concurrent agents, yet quite verbose, which we exploited to verify communication properties of TA [36].

Petri nets come in many flavours, and provide a compact representation of a global model of interaction that avoids the explosion of states caused by the interleaving of independent actions. In [52], a subclass of TA—non-state-sharing vector team automata—has been encoded into *Individual Token Net Controllers*—a model of vector-labeled Petri nets—covering TA in which the synchronisation of a set of agents cannot be influenced by the remaining agents (cf. Sect. 3.3). *Zero-safe nets* are another extension of Petri nets with a transactional mechanism that distinguishes observable from hidden states, used to formalise Reo [74]. This extension could also be used to model TA's synchronisation mechanism, although the analysis of these nets is non-trivial.



MSC (Message Sequence Charts) are visual diagrams commonly used to describe scenarios with interacting agents which have historically been used to describe telecommunication protocols. They are not always precise, and can be enriched with constructs to denote loops, choices, and parallel threads. On the right, we include an informal MSC that captures our Race example, using a loop and a parallel block. Katoen and Lambert [98] have used *pomsets* to formalise MSC, where each possible trace is described as a (multi-)set of actions with a partial order. Guanciale and Tuosto used a pomset semantics for choreographies to reason over realisability [87], and we extended pomsets with a hierarchical structure [80], reasoned over realisability, and compared it to event structures [115] (often used to give semantics to Petri nets). How to use a pomset variation to represent global models for TA is currently being investigated.

I/O automata and related formalisms like I/O systems [97], interface automata [77], reactive transition systems [70], interacting state machines [116], and component-interaction (CI) automata [67] all distinguish input, output, and internal actions. However, I/O automata are *input-enabled*: in every state of the automaton every input action of the automaton is enabled (i.e., executable). In fact, team automata are a generalisation of I/O automata [50]. All formalisms define composition as the synchronous product of automata except for interface automata [77], which restrict product states to compatible states, and CI automata, which were specifically designed to have this distinguishing feature of team automata. CI automata however restrict communication to binary synchronisation between a pair of input and output actions. Contrary to 'classical' team automata, CI automata use system labels to preserve the information about their communication, a feature which inspired the introduction of extended team automata in [44] (cf. Footnote 1 on page 6).

4 Communication Properties

Compatibility of components is an important issue for systems to guarantee successful (*safe*) communication [13,31,32,46,65,69,71,79], i.e., free from message loss (output actions not accepted as input by some other component) and indefinite waiting (for input to be received in the form of an appropriate output action provided by another component). In [31], we identified representative *synchronisation types* to classify synchronisation policies that are realisable in team automata (e.g., binary, multicast and broadcast communication, synchronous product) in terms of ranges for the number of sending and receiving components participating in synchronisations. Moreover, we provided a generic procedure to derive, for each synchronisation type, requirements for *receptiveness* and

for *responsiveness* of team automata that prevent outputs not being accepted and inputs not being provided, respectively, i.e., guaranteeing safe communication. This allowed us to define a notion of *compatibility* for team automata in terms of their compliance with communication requirements, i.e., receptiveness and responsiveness. A team automaton was said to be *compliant* with a given set of communication requirements if in each of its reachable states, the desired communications can immediately occur; it was said to be *weakly compliant* if the communication can eventually occur after some internal actions have been performed (akin to weak compatibility [28,90] or agreement of lazy request actions [15]). Since communication requirements are derived from synchronisation types, we get a family of compatibility notions indexed by synchronisation types.

We revisited the definition of *safe communication* in terms of receptive- and responsiveness requirements in [44,46], due to limitations of our earlier approach.

First, the assignment of a single synchronisation type to a team automata was deemed too restrictive, so we decided to fine tune the number of synchronising sending and receiving components *per action*. For this purpose we introduced, in [44], *synchronisation type specifications* which assign a synchronisation type individually to each communicating action. As we have seen in Sect. 2, such specifications uniquely determine a team automaton. Any synchronisation type specification generates communication requirements to be satisfied by the team.

Receptiveness. Here is the idea. If, in a reachable state q of $\mathbf{ta}(\mathcal{S},\mathbf{st})$, a group $\{\mathcal{A}_n \mid n \in \mathsf{out}\}$ of CA with $\varnothing \neq \mathsf{out} \subseteq \mathcal{N}$ is (locally) enabled to perform an output action a, i.e., for all $n \in \mathsf{out}$ holds $a \in \Sigma_n^!$ and $q_n \xrightarrow{a}_{\mathcal{A}_n}$, and if moreover both (1) the number of CA in out fits that of the senders allowed by the synchronisation type $\mathbf{st}(a) = (O, I)$, i.e., $|\mathsf{out}| \in O$, and (2) the CA need at least one receiver to join the communication, i.e., $0 \notin I$, then we get a *receptiveness requirement*, denoted by $\mathbf{rcp}(\mathsf{out}, a)@q$. If $\mathsf{out} = \{n\}$, we write $\mathbf{rcp}(n, a)@q$ for $\mathbf{rcp}(\{n\}, a)@q$.

Responsiveness. For input actions, one can formulate responsiveness requirements with the idea that enabled inputs should be served by appropriate outputs. The expression $\mathbf{rsp}(\mathsf{in}, a)@q$ is a *responsiveness requirement* if $q \in \mathcal{R}(\mathbf{ta}(\mathcal{S},\mathbf{st}))$, for all $n \in \mathsf{in}$ we have $a \in \Sigma_n^?$ and $q_n \xrightarrow{a}_{\mathcal{A}_n}$, and $|\mathsf{in}| \in I, 0 \notin O$ for $\mathbf{st}(a) = (O, I)$.

Second, we realised that even the weak compliance notion is too restrictive for practical applications. So, in [44], we introduced a much more liberal notion.

Compliance. The TA $\mathbf{ta}(\mathcal{S},\mathbf{st})$ is compliant with a receptiveness requirement $\mathbf{rcp}(\mathsf{out}, a)@q$ if the group of components (with names in out) can find partners in the team which synchronise with the group by taking (receiving) a as input. If reception is immediate, then we speak of receptiveness; if the other components may still perform *arbitrary intermediate actions* (i.e., not limited to internal ones) before accepting a, then we speak of weak receptiveness. We now formally define (weak) compliance, (weak) receptiveness and (weak) responsiveness.

The TA $\mathbf{ta}(\mathcal{S},\mathbf{st})$ is *compliant* with $\mathbf{rcp}(\mathsf{out}, a)@q$ if $\exists_{\mathsf{in}} \cdot q \xrightarrow{(\mathsf{out},a,\mathsf{in})}_{\mathbf{ta}(\mathcal{S},\mathbf{st})}$ while it is *weakly compliant* with $\mathbf{rcp}(\mathsf{out}, a)@q$ if $\exists_{\mathsf{in}} \cdot q \xrightarrow{(\Lambda(\mathbf{st})\setminus_{\mathsf{out}})^* \,;\, (\mathsf{out},a,\mathsf{in})}_{\mathbf{ta}(\mathcal{S},\mathbf{st})}$,

where $\Lambda(\mathbf{st})_{\setminus \mathsf{out}}$ denotes the set of team labels in which no component of out participates. Formally, $\Lambda(\mathcal{S},\mathbf{st})_{\setminus \mathsf{out}} = \{\, (\mathsf{out}', a, \mathsf{in}) \in \Lambda(\mathcal{S},\mathbf{st}) \mid (\mathsf{out}' \cup \mathsf{in}) \cap \mathsf{out} = \varnothing \,\} \cup \{\, (n,a) \in \Lambda(\mathcal{S},\mathbf{st}) \mid n \notin \mathsf{out} \,\}$.

The TA $\mathbf{ta}(\mathcal{S},\mathbf{st})$ is *compliant* with $\mathbf{rsp}(\mathsf{in},a)@q$ if $\exists_{\mathsf{out}} \cdot q \xrightarrow{(\mathsf{out},a,\mathsf{in})}_{\mathbf{ta}(\mathcal{S},\mathbf{st})}$, while it is *weakly compliant* with $\mathbf{rsp}(\mathsf{in},a)@q$ if $\exists_{\mathsf{out}} \cdot q \xrightarrow{(\Lambda(\mathbf{st})_{\setminus \mathsf{in}})^* ; (\mathsf{out},a,\mathsf{in})}_{\mathbf{ta}(\mathcal{S},\mathbf{st})}$, where $\mathbf{st}(\Lambda)_{\setminus \mathsf{in}} = \{(\mathsf{out},a,\mathsf{in}') \in \mathbf{st}(\Lambda) \mid (\mathsf{out} \cup \mathsf{in}') \cap \mathsf{in} = \varnothing\} \cup \{(n,a) \in \mathbf{st}(\Lambda) \mid n \notin \mathsf{in}\}$ denotes the set of team labels in which no component of in participates.

TA: (Weak) Receptiveness. The TA $\mathbf{ta}(\mathcal{S},\mathbf{st})$ is *(weakly) receptive* if for all reachable states $q \in \mathcal{R}(\mathbf{ta}(\mathcal{S},\mathbf{st}))$, the TA $\mathbf{ta}(\mathcal{S},\mathbf{st})$ is (weakly) compliant with *all* receptiveness requirements $\mathbf{rcp}(\mathsf{out},a)@q$ established for q.

Example 3 (Receptiveness and Compliance). In the initial state $(0,0,0)$ of the Race team (cf. Fig. 3), there is a receptiveness requirement of the controller who wants to start the competition, expressed by $\mathbf{rcp}(\mathsf{Ctrl},\mathsf{start})@(0,0,0)$ The TA $\mathbf{ta}(\mathsf{Race},\mathbf{st}_{\mathsf{Race}})$ is compliant with this requirement. When the first runner is in state 2, the desire to send finish leads to three receptiveness requirements: $\mathbf{rcp}(\mathsf{R1},\mathsf{finish})@(1,2,1)$, $\mathbf{rcp}(\mathsf{R1},\mathsf{finish})@(1,2,2)$, and $\mathbf{rcp}(\mathsf{R1},\mathsf{finish})@(2,2,0)$. If the second runner is in state 2, we get three more receptiveness requirements: $\mathbf{rcp}(\mathsf{R2},\mathsf{finish})@(1,1,2)$, $\mathbf{rcp}(\mathsf{R2},\mathsf{finish})@(1,2,2)$, and $\mathbf{rcp}(\mathsf{R2},\mathsf{finish})@(2,0,2)$. The TA $\mathbf{ta}(\mathsf{Race},\mathbf{st}_{\mathsf{Race}})$ is compliant also with these. ▷

Unlike output actions, the selection of an input action of a component is not controlled by the component but by the environment, i.e., there is an external choice. If, for a choice of enabled inputs $\{a_1,\ldots,a_n\}$, *only one of them* can be supplied with a corresponding output of the environment this suffices to guarantee progress of each component waiting for input.

TA: (Weak) Responsiveness. The TA $\mathbf{ta}(\mathcal{S},\mathbf{st})$ is *(weakly) responsive* if for all reachable states $q \in \mathcal{R}(\mathbf{ta}(\mathcal{S},\mathbf{st}))$ and for all $n \in \mathcal{N}$ the following holds: if there is a responsiveness requirement $\mathbf{rsp}(\mathsf{in},a)@q$ established for q with $n \in \mathsf{in}$, then the TA $\mathbf{ta}(\mathcal{S},\mathbf{st})$ is (weakly) compliant with at least one of these requirements.[2]

Example 4 (Responsiveness and Compliance). In the initial state $(0,0,0)$ of the Race team, there is a responsiveness requirement of the two runners who want the competition to start, expressed by $\mathbf{rsp}(\{\mathsf{R1},\mathsf{R2}\},\mathsf{start})@(0,0,0)$. The TA $\mathbf{ta}(\mathsf{Race},\mathbf{st}_{\mathsf{Race}})$ is compliant with this requirement. When the controller is in state 1, there are responsiveness requirements $\mathbf{rsp}(\mathsf{Ctrl},\mathsf{finish})@(1,q_1,q_2)$ for any $q_1,q_2 \in \{1,2\}$. In state $(1,1,1)$, at least one run must happen before a finish is sent; in all other cases, this requirement is immediately fulfilled. Hence, the TA $\mathbf{ta}(\mathsf{Race},\mathbf{st}_{\mathsf{Race}})$ is weakly compliant. There are four more responsiveness requirements when the controller is in state 2, two of which are only weakly fulfilled. ▷

[2] This version of (weak) responsiveness is slightly stronger than in our previous work, driven by the comparison with local deadlock-freedom below.

As far as we know, such powerful compliance notions for I/O-based, synchronous component systems were not studied before. In case of open systems the arbitrary immediate actions before a desired communication happens may be output or input actions open to the environment. Then local communication properties could be violated upon composition with other team automata. This led us to consider *composition* of open team automata and to investigate conditions ensuring *compositionality* of communication properties [33,44,45].

Roadmap. A third limitation has so far not been tackled. In [46], we argued that it may be the case that (local) enabledness of an action indicates only readiness for communication and not so much that communication is required. Therefore, to make this distinction between possible and required communication explicit, we proposed to add designated *final states* to components, where execution can stop but may also continue, in addition to states where progress is required. The addition of final states to component automata has significant consequences for the derivation of communication requirements and for our compliance notions, which would have to be adjusted accordingly.

Verification and Tool Support. Automatically verifying communication properties is non-trivial, as it may involve traversing networks of interacting automata with large state spaces. We pursued a different approach by providing a *logical* characterisation of receptiveness and responsiveness in terms of formulas of a (test-free) propositional dynamic logic [88] using (complex) interactions as modalities (cf. [35,36]). Verification of communication properties then relies on model checking receptive- and responsiveness formulas against a system of component automata taking into account a given synchronisation type specification.

We developed an open-source prototypical tool [56] to support our theory. It implements a transformation of CA, systems, and TA into mCRL2 [5] processes and of the characterising dynamic logic formulas into μ-calculus formulas. The latter is straightforward, whereas the former uses mCRL2's allow operator to suitably restrict the number of multi-action synchronisations such that the semantics of systems of CA is preserved. Then we can automatically check communication properties with the model-checking facilities offered by mCRL2, which outputs the result of the formula as well as a witness or counterexample.

Related Work. The genericity of our approach w.r.t. synchronisation policies allows us to capture compatibility notions for various multi-component coordination strategies. In the literature, compatibility notions are mostly considered for systems relying on peer-to-peer communication, i.e., all synchronisation types are ([1,1],[1,1]). Our notion of receptiveness is inspired by the compatibility notion of interface automata [77] and indeed both notions coincide for closed systems and 1-to-1 communication. It also coincides with receptiveness in [70]. Weak receptiveness is inspired by the notion of weak compatibility in [28] and also corresponds to unspecified reception in the context of n-protocol compatibility in [79] and lazy request in contract automata [15]. We are not aware of compatibility notions concerning responsiveness. In [70], it is captured by

deadlock-freedom and in [79] it is expressed by part of the definition of bidirectional complementary compatibility which, however, does not support choice of inputs as we do.

The relationship between deadlock-freedom, used in different variations in the literature, and our communication properties is subtle. Note that the distinction between input and output actions is not relevant for the deadlock notions and that two types of deadlocks are often distinguished: global and local deadlocks (cf., e.g., [6]). For the following discussion, we assume that the components of a system have no final state (i.e., for each local state there is an outgoing transition). We further assume that all local actions are external, i.e., input or output, and that the synchronisation types of all output actions are ([1,1],[1,*]) and ([1,*],[1,1]) for all input actions. Then weak receptiveness together with weak responsiveness of team automata implies (global) deadlock-freedom in the sense of BIP [84] and the equivalent notion of stuck-freeness in MPST [126]. The weaker notion of deadlock-freedom in ChorAut [10] is also implied by the combination of weak receptiveness with weak responsiveness.

Global deadlock-freedom does, however, not imply weak receptiveness. For instance, assume that there are two CA $\mathcal{A}_1$ and $\mathcal{A}_2$ such that in the initial state q_1 of $\mathcal{A}_1$ there is a choice of two outputs a and b, and in the initial state q_2 of $\mathcal{A}_2$ there is only one outgoing transition with input a. Then the system state (q_1, q_2) is not a deadlock state but the receptiveness requirement for b is violated at state (q_1, q_2) since the autonomous choice of output b by the first component would not be accepted by the second. Also weak responsiveness is not implied by global deadlock-freedom. For a counterexample we would need three components, two with a single input, say a for the first and b for the second, and one with a single output, say a. Assume that all components have loops around the initial state. Then the system, considered under 1-to-1 communication, would be (globally) deadlock-free but not (weakly) responsive, since the second component would never receive b.

This example points out the crucial difference between global and local deadlocks, covered by the notions of "individual deadlock" in BIP [84], "lock" in ChorAut [10], and "strong lock" in MPST [126]. The difference between the latter two is that [10] assumes fair runs. The notion of individual deadlock in [84] is defined differently, but seems equivalent to a "lock" in [10] for 1-to-1 communication. Weak receptiveness together with weak responsiveness is equivalent to individual deadlock-freedom in BIP if the interaction model fits to the synchronisation types assumed above. Hence, this is also equivalent to lock-freedom in [10]. Strong lock-freedom in MPST [126] is indeed a stronger requirement.

The compatibility notions above formalise general requirements for safe communication. Some approaches prefer to formulate individual compatibility requirements tailored to particular applications by formulas in a logic for dynamic systems using model-checking tools for verification (cf., e.g., [3,101, 103,118]).

5 Realisability

In this section, we consider a top-down method where first a global model $\mathcal{M}$ for the intended interaction behaviour of a system is provided on the basis of a given system signature and synchronisation type specification (STS) $\mathbf{st}$. Then our goal is to construct a system $\mathcal{S}$ of component automata from which a team automaton $\mathbf{ta}(\mathcal{S}, \mathbf{st})$ can be generated that complies with the global model $\mathcal{M}$.

For this purpose, we instantiate the generic approach investigated in [47]. This instantiation applies the localisation style with so-called "poor" local actions of the form $!a$ for outputs and $?a$ for inputs; localisations with "rich" local actions, which mention in the local context of a component n the name of a receiver m of a message (e.g., by $nm!a$) or the sender m of a message (e.g., by $mn?a$), are treated in [47] as well, but they are not relevant for team automata.

We assume the notions of component automaton (CA), system $\mathcal{S}$, synchronisation type specification $\mathbf{st}$, and generated team automaton $\mathbf{ta}(\mathcal{S}, \mathbf{st})$, as provided in Sect. 2. Our realisation method is summarised in Fig. 10 and will be explained in more detail in the next sections.

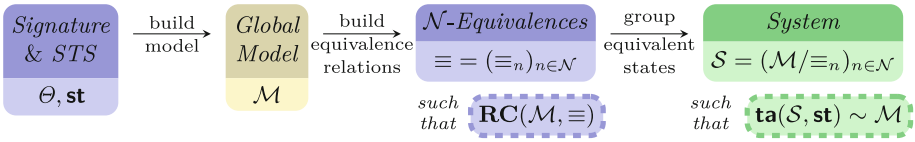


Fig. 10. Realisation method

5.1 Global Models of Interaction

Our method starts with a *system signature* $\Theta = (\mathcal{N}, (\Sigma_n)_{n\in\mathcal{N}})$, where $\mathcal{N}$ is a finite, nonempty set of component names and $(\Sigma_n)_{n\in\mathcal{N}}$ is an $\mathcal{N}$-indexed family of action sets $\Sigma_n = \Sigma_n^? \uplus \Sigma_n^!$ split into disjoint sets $\Sigma_n^?$ of *input actions* and $\Sigma_n^!$ of *output actions*. As in Sect. 2, let $\Sigma^\bullet = \bigcup_{n\in\mathcal{N}} \Sigma_n^? \cap \bigcup_{n\in\mathcal{N}} \Sigma_n^!$ be the set of *communicating actions*. We do not consider internal actions here and we assume that all system actions $a \in \bigcup_{n\in\mathcal{N}} \Sigma_n$ are communicating.

Together with the system signature a synchronisation type specification $\mathbf{st}$ must be provided assigning to each $a \in \Sigma^\bullet$ a pair of intervals $\mathbf{st}(a) = (O, I)$, as explained in Sect. 2. The system signature Θ together with the STS $\mathbf{st}$ determine the following set $\Lambda(\Theta, \mathbf{st})$ of *(multi-)interactions* respecting the constraints of the given synchronisation types:

$$\Lambda(\Theta, \mathbf{st}) = \{\, (\mathsf{out}, a, \mathsf{in}) \mid \varnothing \neq (\mathsf{out} \cup \mathsf{in}) \subseteq \mathcal{N},\ \forall_{n\in\mathsf{out}} \cdot a \in \Sigma_n^!,\ \forall_{n\in\mathsf{in}} \cdot a \in \Sigma_n^?,\ \mathbf{st}(a) = (O, I) \Rightarrow |\mathsf{out}| \in O \wedge |\mathsf{in}| \in I \,\}$$

We model the global interaction behaviour of the intended system by an LTS whose labels are (multi-)interactions in $\Lambda(\Theta, \mathbf{st})$. Hence, a *global interaction model* over Θ and $\mathbf{st}$ is an LTS of the form $\mathcal{M} = (Q, q_0, \Lambda(\Theta, \mathbf{st}), E)$.

Example 5. To develop the Race system we would start with system signature Θ_{Race} with component names Ctrl, R1, R2 and action sets $\Sigma^{?}_{\mathsf{Ctrl}} = \{\mathsf{finish}\} = \Sigma^{!}_{\mathsf{R1}} = \Sigma^{!}_{\mathsf{R2}}$ and $\Sigma^{!}_{\mathsf{Ctrl}} = \{\mathsf{start}\} = \Sigma^{?}_{\mathsf{R1}} = \Sigma^{?}_{\mathsf{R2}}$. We do not consider the internal run action. As in Example 2, we use the STS $\mathbf{st}_{\mathsf{Race}}$ with $\mathbf{st}_{\mathsf{Race}}(\mathsf{start}) = ([1,1],[2,2])$ and $\mathbf{st}_{\mathsf{Race}}(\mathsf{finish}) = ([1,1],[1,1])$. Figure 11 shows on the left the induced interaction set $\Lambda(\Theta_{\mathsf{Race}}, \mathbf{st}_{\mathsf{Race}})$ (abbreviated by Λ_{Race}) and on the right a global interaction model $\mathcal{M}_{\mathsf{Race}}$. It imposes the system to start in a (global) state, where the controller starts both runners at once. Each runner separately sends a finish signal to the controller (in arbitrary order). After that a new run can start. ▷

$$\Lambda_{\mathsf{Race}} = \left\{ \begin{array}{l} \mathsf{Ctrl} \to \{\mathsf{R1}, \mathsf{R2}\} : \mathsf{start}\,, \\ \mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}\,, \\ \mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish} \end{array} \right\}$$

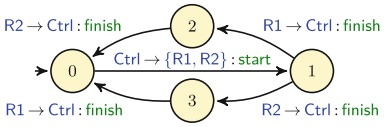


Fig. 11. Interaction set Λ_{Race} and global interaction model $\mathcal{M}_{\mathsf{Race}}$

Remark 1. Our development methodology can be extended by providing first an abstract specification of desired and forbidden interaction properties from a global perspective. In [47], we proposed a propositional dynamic logic for this purpose where interactions are used as atomic actions in modalities such that we can express safety and (a kind of) liveness properties. For instance, we could express the following requirements for the Race system by dynamic logic formulas, using the usual box modalities $[\cdot]$ and $\langle\cdot\rangle$, sequential composition $(;)$, choice $(+)$, and iteration $(\cdot^*)$:

1. *"For any started runner, it should be possible to finish her/his run."*
$$\Big[some^*; \mathsf{Ctrl} \to \{\mathsf{R1}, \mathsf{R2}\} : \mathsf{start}\Big] \left(\begin{array}{l} \langle some^*; \mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}\rangle\, true \wedge \\ \langle some^*; \mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish}\rangle\, true \end{array} \right)$$
2. *"No runner should finish before she/he was started by the controller."*
$$\left[\Big(- (\mathsf{Ctrl} \to \{\mathsf{R1}, \mathsf{R2}\} : \mathsf{start}) \Big)^* \; ; \; \left(\begin{array}{l} \mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish} \; + \\ \mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish} \end{array} \right) \right] \mathit{false}$$

▷

To check that a global interaction model satisfies a specification, we propose to use the mCRL2 toolset [5,85]. For this purpose, as explained in [36], we can use the translation from LTS models into process expressions and the translation from our interaction-based dynamic logic into the syntax used by mCRL2.

5.2 Realisation of Global Models of Interaction

Our central task concerns the realisation (decomposition) of a global interaction model $\mathcal{M}$ in terms of a (possibly distributed) system $\mathcal{S}$ of component automata which are coordinated according to the given synchronisation type specification.

In order to formulate our realisation notion, we briefly recall the standard notion of bisimulation. Let $\mathcal{L}_n = (Q_n, q_{n,0}, \Sigma, E_n)$ be two LTS (for $n = 1, 2$) over the same action set Σ. A *bisimulation relation* between $\mathcal{L}_1$ and $\mathcal{L}_2$ is a relation $B \subseteq Q_1 \times Q_2$ such that for all $(q_1, q_2) \in B$ and for all $a \in \Sigma$ the following holds:

1. if $q_1 \xrightarrow{a}_{\mathcal{L}_1} q_1'$, then there exist $q_2' \in Q_2$ and $q_2 \xrightarrow{a}_{\mathcal{L}_2} q_2'$ such that $(q_1', q_2') \in B$;
2. if $q_2 \xrightarrow{a}_{\mathcal{L}_2} q_2'$, then there exist $q_1' \in Q_1$ and $q_1 \xrightarrow{a}_{\mathcal{L}_1} q_1'$ such that $(q_1', q_2') \in B$.

$\mathcal{L}_1$ and $\mathcal{L}_2$ are *bisimilar*, denoted by $\mathcal{L}_1 \sim \mathcal{L}_2$, if there exists a bisimulation relation B between $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $(q_{1,0}, q_{2,0}) \in B$.

Now, we assume given a system signature $\Theta = (\mathcal{N}, (\Sigma_n)_{n\in\mathcal{N}})$, an STS $\mathbf{st}$ and a global interaction model $\mathcal{M}$ with labels $\Lambda(\Theta, \mathbf{st})$. A system $\mathcal{S} = (\mathcal{N}, (\mathcal{A}_n)_{n\in\mathcal{N}})$ of component automata $\mathcal{A}_n$ with actions Σ_n is a *realisation* of $\mathcal{M}$ w.r.t. $\mathbf{st}$ if the team automaton $\mathbf{ta}(\mathcal{S}, \mathbf{st})$ generated over $\mathcal{S}$ by $\mathbf{st}$ (as defined in Sect. 2) is bisimilar to $\mathcal{M}$, i.e., $\mathbf{ta}(\mathcal{S}, \mathbf{st}) \sim \mathcal{M}$. We note that the team labels $\Lambda(\mathcal{S}, \mathbf{st})$ of $\mathbf{ta}(\mathcal{S}, \mathbf{st})$ are exactly the interactions in $\Lambda(\Theta, \mathbf{st})$, i.e., the actions of the LTS $\mathcal{M}$. The global model $\mathcal{M}$ is called *realisable* if such a system $\mathcal{S}$ exists.

Remark 2. Technically, the definition of realisability in [47] uses a synchronous Γ-composition $\otimes_\Gamma(\mathcal{A}_n)_{n\in\mathcal{N}}$ [47, Def. 7] of the component automata. Transferred to the context of synchronisation types, Γ would be $\Lambda(\Theta, \mathbf{st})$. Moreover, $\otimes_\Gamma(\mathcal{A}_n)_{n\in\mathcal{N}}$ contains only reachable states, which need not to be the case for the team automaton $\mathbf{ta}(\mathcal{S}, \mathbf{st})$. However, we can restrict $\mathbf{ta}(\mathcal{S}, \mathbf{st})$ to its reachable sub-LTS which coincides with $\otimes_\Gamma(\mathcal{A}_n)_{n\in\mathcal{N}}$. Note also that any LTS is bisimilar to its reachable sub-LTS, such that this restriction does not harm. ▷

Since our realisability notion relies on bisimulation, we are able to deal with non-deterministic behaviour. Note that, according to the invariance of propositional dynamic logic under bisimulation (cf. [59]), we obtain that global models and their realisations satisfy the same propositional dynamic logic formulas when (multi-)interactions are used as atomic actions as proposed in [36,47].

Next, we consider the following two important questions:

1. How can we check whether a given global model $\mathcal{M}$ is realisable?
2. If it is, how can we build/synthesise a concrete realisation?

To tackle the first question, we propose to find a family $\equiv = (\equiv_n)_{n\in\mathcal{N}}$ of equivalence relations on the *global* state space Q of $\mathcal{M}$ such that, for each component name $n \in \mathcal{N}$ and states $q, q' \in Q$, $q \equiv_n q'$ expresses that q and q' are indistinguishable from the viewpoint of i. It is required that at least any two global states $q, q' \in Q$ which are related by a global transition $q \xrightarrow{(\mathsf{out},a,\mathsf{in})}_{\mathcal{M}} q'$ should be indistinguishable for any $i \in \mathcal{N}$ which does not participate in the interaction, i.e., $q \equiv_n q'$ for all $n \notin \mathsf{out} \cup \mathsf{in}$. A family $\equiv = (\equiv_n)_{n\in\mathcal{N}}$ of equivalence relations $\equiv_n \subseteq Q \times Q$ with this property is called an $\mathcal{N}$-*equivalence*.

We can now formulate our realisability condition for the global model $\mathcal{M} = (Q, q_0, \Lambda(\Theta, \mathbf{st}), E)$. Our goal is to find an $\mathcal{N}$-equivalence $\equiv = (\equiv_n)_{n\in\mathcal{N}}$ over $\mathcal{M}$ such that the following *realisability condition* $\mathbf{RC}(\mathcal{M}, \equiv)$ holds.

RC$(\mathcal{M}, \equiv)$**:** For each interaction $(\mathsf{out}, a, \mathsf{in}) \in \Lambda(\Theta, \mathbf{st})$, whenever there is (1) a map $\ell : \mathsf{out} \cup \mathsf{in} \to Q$ assigning a global state $q_n = \ell(n)$ to each $n \in \mathsf{out} \cup \mathsf{in}$ together with (2) a global *"glue"* state g, i.e., $q_n \equiv_n g$ for each $n \in \mathsf{out} \cup \mathsf{in}$, then we expect: for all $n \in \mathsf{out} \cup \mathsf{in}$ and global transitions $q_n \xrightarrow{(\mathsf{out}_n, a, \mathsf{in}_n)}_{\mathcal{M}} q'_n$ in which n participates (i.e., $n \in \mathsf{out}_n \cap \mathsf{in}_n$), there is be a global transition $g \xrightarrow{(\mathsf{out}, a, \mathsf{in})}_{\mathcal{M}} g'$ such that $q'_n \equiv_n g'$ for each $n \in \mathsf{out} \cup \mathsf{in}$.

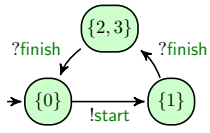
The intuition for this requirement is that if component n can participate in executing an action a in state q_n, then n should also be able to participate in executing a in state g, since n cannot distinguish q_n and g. Because this should hold for all $n \in \mathsf{out} \cup \mathsf{in}$, the interaction $(\mathsf{out}, a, \mathsf{in})$ should be enabled in g and preserve indistinguishability of states for all $n \in \mathsf{out} \cup \mathsf{in}$.

As a consequence of our results in [47], in particular Thm. 3, we obtain that if there is an $\mathcal{N}$-equivalence $\equiv = (\equiv_n)_{n\in\mathcal{N}}$ such that the realisability condition $\mathbf{RC}(\mathcal{M}, \equiv)$ holds, then the global model $\mathcal{M} = (Q, q_0, \Lambda(\Theta, \mathbf{st}), E)$ can indeed be realised by the system $\mathcal{S}_\equiv = (\mathcal{M}/\equiv_n)_{n\in\mathcal{N}}$ of component automata constructed as local quotients of $\mathcal{M}$, i.e., $\mathbf{ta}(\mathcal{S}_\equiv, \mathbf{st}) \sim \mathcal{M}$. More precisely, the *local n-quotient* of $\mathcal{M}$ is the component automaton $\mathcal{M}/\equiv_n \; = (Q/\equiv_n, [q_0]_{\equiv_n}, \Sigma_n, (E/\equiv_n)$, where

- $Q/\equiv_n \; = \{\, [q]_{\equiv_n} \mid q \in Q \,\}$,
- $E/\equiv_n$ is the least set of transitions generated by the rule

$$\frac{q \xrightarrow{(\mathsf{out}, a, \mathsf{in})}_{\mathcal{M}} q' \qquad n \in \mathsf{out} \cup \mathsf{in}}{[q]_{\equiv_n} \xrightarrow{\mathsf{a}}_{(\mathcal{M}/\equiv_n)} [q']_{\equiv_n}}$$

Example 6. Take the global LTS $\mathcal{M}_{\mathsf{Race}}$ in Fig. 11. We use the family of equivalences $\equiv = (\equiv_n)_{n\in\{\mathsf{Ctrl},\mathsf{R1},\mathsf{R2}\}}$ that obeys $\mathbf{RC}(\mathcal{M}_{\mathsf{Race}}, \equiv)$ (see below) and partitions the state space Q in $Q/\equiv_{\mathsf{Ctrl}} = \{\{0\}, \{1\}, \{2, 3\}\}$, $Q/\equiv_{\mathsf{R1}} = \{\{0, 2\}, \{1, 3\}\}$, and $Q/\equiv_{\mathsf{R2}} = \{\{0, 3\}, \{1, 2\}\}$. Using these equivalences, the local quotients are:

$(\mathcal{M}/\equiv_{\mathsf{Ctrl}}) =$ 

$(\mathcal{M}/\equiv_{\mathsf{R1}}) =$ 

$(\mathcal{M}/\equiv_{\mathsf{R2}}) =$ 

So we obtained a system that is a realisation of $\mathcal{M}_{\mathsf{Race}}$. This means the team automaton $\mathbf{ta}(\mathcal{S}_\equiv, \mathbf{st}_{\mathsf{Race}})$ generated by $\mathcal{S}_\equiv$ and $\mathbf{st}_{\mathsf{Race}}$ is bisimilar to $\mathcal{M}_{\mathsf{Race}}$. Indeed, both are the same LTS up to renaming of states: state $(\{0\}, \{0, 2\}, \{0, 3\})$ in $\mathbf{ta}(\mathcal{S}_\equiv, \mathbf{st}_{\mathsf{Race}})$ instead of state 0 in $\mathcal{M}_{\mathsf{Race}}$, $(\{1\}, \{1, 3\}, \{1, 2\})$ instead of 1, $(\{2, 3\}, \{0, 2\}, \{1, 2\})$ instead of 2, and $(\{2, 3\}, \{1, 3\}, \{0, 3\})$ instead of 3.

We now show how to check $\mathbf{RC}(\mathcal{M}_{\mathsf{Race}}, \equiv)$ using interaction $\mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}$ as example. We have $1 \xrightarrow{\mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}}_{\mathcal{M}_{\mathsf{Race}}} 2$ and $1 \xrightarrow{\mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish}}_{\mathcal{M}_{\mathsf{Race}}} 3$. Taking

1 as a (trivial) glue state, we thus have, as required, $1 \xrightarrow{\mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}}_{\mathcal{M}_{\mathsf{Race}}} 2$, *but also* it is required that $2 \equiv_{\mathsf{Ctrl}} 3$ must hold, which is the case. Note that we would not have succeeded here if we had taken the identity for $\equiv_{\mathsf{Ctrl}}$. ▷

Tool Support. We implemented a prototypical tool, called Ceta, to perform realisability checks and system synthesis (cf. [47,48]). It is open source, available at https://github.com/arcalab/choreo/tree/ceta, and executable by navigating to https://lmf.di.uminho.pt/ceta. It provides a web browser where one can input a global protocol described in a choreographic language, resembling regular expressions of interactions. It offers automatic visualisation of the protocol as a state machine representing a global model and it includes examples with descriptions.

Ceta implements a *constructive* approach to the *declarative* description of the realisability conditions. It builds a family of equivalence relations, starting with one that groups states connected by transitions in which the associate participant is not involved. This family of minimal equivalence relations is checked for satisfaction of the realisability condition w.r.t. the global model. If it fails, a new attempt is started after extending the equivalence relations appropriately. If no failure occurs, then the realisability condition is satisfied and the resulting equivalence classes are used to join equivalent states in the global model, yielding local quotients which can again be visualised. Thus a realisation of the global model is constructed. There are several widgets that provide further insights, such as the intermediate equivalence classes, the synchronous composition of local components, and bisimulations between global models and team automata. Readable error messages are given when a realisability condition does not hold.

5.3 Related Work

Our approach is driven by specified sets of multi-interactions supporting any kind of synchronous communication between multiple senders and multiple receivers. To the best of our knowledge, realisations of global models with arbitrary multi-interactions have not yet been studied in the literature. There are, however, specialised approaches that deal with realisations of global models or decomposition of transition systems. In this section, we first provide a revised and extended comparison of our approach with that of Castellani et al. [73], followed by a brief comparison with other approaches.

Relationship to [73]. Our realisability condition $\mathbf{RC}(\mathcal{M}, \equiv)$, based on the notion of $\mathcal{N}$-equivalence $\equiv$, is strongly related to a condition for implementability in [73, Theorem 3.1]. The main differences are:

1. In [73], there is no distinction between input and output actions.

2. In [73], interactions are always full synchronisations on an action a, while we deal with individual multi-interactions $(\mathsf{out}, a, \mathsf{in})$ specified by an STS. Of course, we can also use an STS $\mathbf{st}_{full}$ for full synchronisation. Then we define, for each action a, $\mathbf{st}_{full}(a) = ([\#out(a), \#out(a)], [\#in(a), \#in(a)])$, where $\#out(a) = |\{n \in \mathcal{N} \mid a \in \Sigma_n^!\}|$ is the number of components having a as an output action and $\#in(a) = |\{n \in \mathcal{N} \mid a \in \Sigma_n^?\}|$ is the number of components having a as an input action.
3. In [73], they provide a characterisation of implementability up to isomorphism, while we provide a criterion for realisability modulo bisimulation, thus supporting non-determinism. To achieve this, we basically omitted condition (ii) of [73, Theorem 3.1] which requires that whenever two global states q and q' are n-equivalent for all $n \in \mathcal{N}$, then $q = q'$. In [48, Example 8], we provide a simple example for a global interaction model which satisfies our realisability condition but for which there is no realisation up to isomorphism. Note that [73, Theorem 6.2] provides a proposal to deal with a characterisation of implementability modulo bisimulation under the assumption of deterministic product transition systems. The authors also report on a result to characterise implementability for non-deterministic product systems which uses infinite execution trees and is thus, unfortunately, not effective.

Relationship to Other Approaches

1. Our correctness notion for realisation of global models by systems of communicating local components is based on bisimulation, beyond language-based approaches like [11,72] with realisability expressed by language equivalence.
2. For realisable global models, we construct realisations in terms of systems of local quotients. This technique differs from projections used, e.g., in the field of MPST, where projections are partial operations depending on syntactic conditions (cf., e.g., [58]). In our approach, no restrictions on the form of global models are assumed. On the other hand, the syntactic restrictions used for global types guarantee some kind of communication properties of a resulting system which we consider separately (cf. Sect. 4).
3. There are other papers in the literature in the context of different formalisms dealing with decomposition of port automata [102], Petri nets, or algebraic processes into (indecomposable) components [109,112] used for the efficient verification and parallelisation of concurrent systems [75,86] or to obtain better (optimised) implementations [131,132].

Roadmap

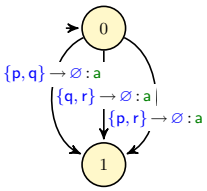
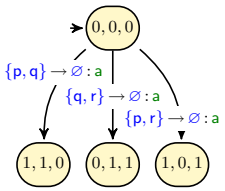
1. Our current realisability approach does not deal with internal actions, which are however also an ingredient of the team automata framework and represented by system labels of the form (n, a) (cf. Sect. 2). They naturally appear when we build a team of CA which have internal actions. We believe, however, that internal actions should not be part of a global interaction model whose purpose is to present the *observable* interaction behaviour of an intended system. To bridge the gap, the idea is to relax the realisation notion by requiring only a weak bisimulation relation between a global model and the team automaton of a system realisation with internal actions.
2. Another aspect concerns the fact that, in general, it may happen that a global interaction model does not satisfy the realisability condition but is nevertheless realisable. Therefore, we want to look for a weaker version of our realisability condition making it necessary and sufficient for realisations based on bisimulation. The following example shows that our current realisability condition is only sufficient.

Example 7. Consider the system signature Θ with component names $\mathcal{N} = \{\mathsf{p}, \mathsf{q}, \mathsf{r}\}$ and with the action sets $\Sigma_n^! = \{\mathsf{a}\}, \Sigma_n^? = \varnothing$ for $n \in \mathcal{N}$ and We use the STS with $\mathbf{st}(\mathsf{a}) = ([2,2],[0,0])$. Then the interaction set is $\Lambda(\Theta, \mathbf{st}) = \big\{\{\mathsf{p},\mathsf{q}\} \to \varnothing : \mathsf{a}, \{\mathsf{q},\mathsf{r}\} \to \varnothing : \mathsf{a}, \{\mathsf{p},\mathsf{r}\} \to \varnothing : \mathsf{a}\big\}$. The global model $\mathcal{M}$ in Table 2 (left) is realisable by the system $\mathcal{S} = \{\mathcal{M}_\mathsf{p}, \mathcal{M}_\mathsf{q}, \mathcal{M}_\mathsf{r}\}$, whose CA are shown in Table 2 (middle). To see this, we compute the team automaton $\mathbf{ta}(\mathcal{S}, \mathbf{st})$ shown in Table 2 (right). Obviously, $\mathcal{M}$ and $\mathbf{ta}(\mathcal{S}, \mathbf{st})$ are bisimilar and hence $\mathcal{M}$ is realisable. However, there is no $\mathcal{N}$-equivalence $\equiv$ such that the realisability condition holds. We now prove this by contradiction.

Assume that $\equiv = \{\equiv_\mathsf{p}, \equiv_\mathsf{q}, \equiv_\mathsf{r}\}$ is an $\mathcal{N}$-equivalence such that $\mathbf{RC}(\mathcal{M}, \equiv)$ holds. Now consider the interaction $\{\mathsf{p},\mathsf{q}\} \to \varnothing : \mathsf{a}$, the global state 0 of $\mathcal{M}$ and the transition $0 \xrightarrow{\{\mathsf{p},\mathsf{q}\} \to \varnothing : \mathsf{a}}_{\mathcal{M}} 1$. Obviously, $0 \equiv_\mathsf{p} 1$ and $1 \equiv_\mathsf{q} 0$ must hold because of the transition $0 \xrightarrow{\{\mathsf{q},\mathsf{r}\} \to \varnothing : \mathsf{a}}_{\mathcal{M}} 1$ in which p does not participate and the transition $0 \xrightarrow{\{\mathsf{p},\mathsf{r}\} \to \varnothing : \mathsf{a}}_{\mathcal{M}} 1$ in which q does not participate. So we can take 1 as a glue state between the global states $q_1 = 0$ and $q_2 = 0$. Then we consider the interaction $\{\mathsf{p},\mathsf{q}\} \to \varnothing : \mathsf{a}$ one time for q_1 and one time for q_2. Since we assumed $\mathbf{RC}(\mathcal{M}, \equiv)$, there must be a transition $1 \xrightarrow{\{\mathsf{p},\mathsf{q}\} \to \varnothing : \mathsf{a}}_{\mathcal{M}}$ leaving the glue state, which is not the case. Contradiction! ▷

3. We are interested in a compositional approach to construct larger realisations from smaller ones. Then compositionality of realisability is important.
4. We want to study under which conditions on global models and synchronisation types our communication properties can be guaranteed for realisations.

Table 2. Global $\mathcal{M}$ does not satisfy $\mathbf{RC}(\mathcal{M},\equiv)$, but $\mathcal{S}=\{\mathcal{M}_p, \mathcal{M}_q, \mathcal{M}_r\}$ realises $\mathcal{M}$

global $\mathcal{M}$	local $\mathcal{M}_p$	local $\mathcal{M}_q$	Local $\mathcal{M}_r$	$\mathbf{ta}(\S, \mathbf{st})$
				

6 Further Aspects

Recently, we also proposed featured team automata [34] to support *variability* in the development and analysis of teams, capable of concisely capturing a family of concrete product (automaton) models for specific configurations determined by feature selection, as is common in software product line engineering [1]. Variability has also been studied for several related coordination models, such as BIP [104,111], Contract automata [15,16,19], Reo [120], Petri nets [113,114], and I/O automata [106,108]. We did not address *data*, for which Reo and BIP provide native support, since team automata cannot currently deal with that.

In the past, we studied in detail the computations and behaviour of team automata in relation to that of their constituting component automata [49,51], identifying several types of team automata that satisfy *compositionality* in terms of (synchronised) shuffles of their computations (i.e., formal languages). In [42, 43], a process calculus for modelling team automata was introduced, extending some classical results on I/O automata as well as the family of team automata that guarantee a degree of compositionality. Finally, team automata were used for the analysis of *security* aspects in communication protocols [29,54,55,81], in particular for spatial and spatio-temporal access control [40,94].

7 Conclusion

We provided an overview of team automata, a model for capturing a variety of notions related to coordination in distributed systems of systems with decades of history (cf. Appendix A) as witnessed by 25+ publications by 25+ researchers,[3] and compared them rather informally with related models for coordination (cf. Table 1). A single running example modelled in the various formalisms eases their comparison. We focused on differences in synchronisation and composition, but also addressed communication properties, realisability, verification, and tool support—all aspects we investigated during the last five years. Team automata support very flexible types of synchronous communication. They are

[3] http://fmt.isti.cnr.it/~mtbeek/TA.html.

not designed for asynchronous communication (yet), like asynchronous multiparty session types [92] or systems of communicating finite state machines [66], for which other kinds of compositions and communication properties are relevant [9].

Acknowledgements. We thank Davide Basile, Simon Bliudze and the three anonymous reviewers for their comments and suggestions that helped us to improve this paper.

Maurice ter Beek was funded by MUR PRIN 2020TL3X8X project T-LADIES (Typeful Language Adaptation for Dynamic, Interacting and Evolving Systems) and CNR project "Formal Methods in Software Engineering 2.0", CUP B53C24000720005.

José Proença was supported by the CISTER Research Unit (UIDP/UIDB/04234/2020), financed by National Funds through FCT/MCTES (Portuguese Foundation for Science and Technology), and by project IBEX (PTDC/CCI-COM/4280/2021) financed by national funds through FCT.

A Selected Publications from 25+ Years of Team Automata

Year		Title	Venue
2023	[47]	Realisability of Global Models of Interaction	ICTAC'23
2023	[119]	Overview on Constrained Multiparty Synchronisation in Team Automata	FACS'23
2023	[36]	Can we Communicate? Using Dynamic Logic to Verify Team Automata	FM'23
2021	[34]	Featured Team Automata	FM'21
2020	[46]	Team Automata@Work: On Safe Communication	COORDINATION'20
2020	[44]	Compositionality of Safe Communication in Systems of Team Automata	ICTAC'20
2017	[31]	Communication Requirements for Team Automata	COORDINATION'17
2016	[32]	Conditions for Compatibility of Components: The Case of Masters and Slaves	ISoLA'16
2014	[53]	On Distributed Cooperation and Synchronised Collaboration	JALC
2013	[69]	Compatibility in a multi-component environment	TCS
2012	[52]	Vector Team Automata	TCS
2010	[94]	Team Automata Based Framework for Spatio-Temporal RBAC Model	BAIP'10
2009	[51]	Associativity of Infinite Synchronized Shuffles and Team Automata	Fundam. Inform.
2008	[128]	Extending Team Automata to Evaluate Software Architectural Design	COMPSAC'08

Year		Title	Venue
2008	[43]	A calculus for team automata	ENTCS
2007	[129]	A Review on Specifying Software Architectures Using Extended Automata-Based Models	FSEN'07
2006	[81]	Modelling a Secure Agent with Team Automata	ENTCS
2006	[55]	A Team Automaton Scenario for the Analysis of Security Properties in Communication Protocols	JALC
2005	[54]	Team Automata for Security – A Survey –	ENTCS
2005	[50]	Modularity for Teams of I/O Automata	IPL
2004	[37]	Teams of Pushdown Automata	IJCM
2004	[68]	Interactive Behaviour of Multi-Component Systems	Workshop
2003	[30]	Team Automata: A Formal Approach to the Modeling of Collaboration Between System Components	PhD thesis
2003	[49]	Team Automata Satisfying Compositionality	FME'03
2003	[29]	Team Automata for Security Analysis of Multicast/Broadcast Communication	Workshop
2003	[100]	Team Automata for CSCW – A Survey –	LNCS
2003	[41]	Synchronizations in Team Automata for Groupware Systems	CSCW
2002	[83]	Towards Team-Automata-Driven Object-Oriented Collaborative Work	LNCS
2001	[40]	Team Automata for Spatial Access Control	ECSCW'01
2001	[39]	Team Automata for CSCW	Workshop
2000	[91]	A Conflict-Free Strategy for Team-Based Model Development	Workshop
1999	[38]	Synchronizations in Team Automata for Groupware Systems	Tech. Rep.
1997	[82]	Team Automata for Groupware Systems	GROUP'97

References

1. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37521-7
2. Arbab, F.: Reo: a channel-based coordination model for component composition. Math. Struct. Comput. Sci. **14**(3), 329–366 (2004). https://doi.org/10.1017/S0960129504004153
3. Arbab, F., Kokash, N., Meng, S.: Towards using Reo for compliance-aware business process modeling. In: Margaria, T., Steffen, B. (eds.) ISoLA 2008. CCIS, vol. 17, pp. 108–123. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88479-8_9
4. Arbab, F., Krause, C., Maraikar, Z., Moon, Y.J., Proença, J.: Modeling, testing and executing Reo connectors with the Eclipse coordination tools. Tool demo session of FACS 2008 (2008)
5. Atif, M., Groote, J.F.: Understanding Behaviour of Distributed Systems Using mCRL2. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-23008-0
6. Attie, P.C., Bensalem, S., Bozga, M., Jaber, M., Sifakis, J., Zaraket, F.A.: Global and local deadlock freedom in BIP. ACM Trans. Softw. Eng. Methodol. **26**(3), 9:1–9:48 (2018). https://doi.org/10.1145/3152910
7. Baier, C., Sirjani, M., Arbab, F., Rutten, J.J.M.M.: Modeling component connectors in Reo by constraint automata. Sci. Comput. Program. **61**(2), 75–113 (2006). https://doi.org/10.1016/J.SCICO.2005.10.008

8. Baranov, E., Bliudze, S.: Expressiveness of component-based frameworks: a study of the expressiveness of BIP. Acta Inform. **57**, 761–800 (2020). https://doi.org/10.1007/s00236-019-00337-7
9. Barbanera, F., de'Liguoro, U., Hennicker, R.: Connecting open systems of communicating finite state machines. J. Log. Algebr. Methods Program. **109** (2019). https://doi.org/10.1016/j.jlamp.2019.07.004
10. Barbanera, F., Lanese, I., Tuosto, E.: Choreography automata. In: Bliudze, S., Bocchi, L. (eds.) COORDINATION 2020. LNCS, vol. 12134, pp. 86–106. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-50029-0_6
11. Barbanera, F., Lanese, I., Tuosto, E.: Formal choreographic languages. In: ter Beek, M.H., Sirjani, M. (eds.) COORDINATION 2022. LNCS, vol. 13271, pp. 121–139. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-08143-9_8
12. Barbanera, F., Lanese, I., Tuosto, E.: A theory of formal choreographic languages. Log. Meth. Comp. Sci. **19**(3), 9:1–9:36 (2023). https://doi.org/10.46298/LMCS-19(3:9)2023
13. Bartoletti, M., Cimoli, T., Zunino, R.: Compliance in behavioural contracts: a brief survey. In: Bodei, C., Ferrari, G.-L., Priami, C. (eds.) Programming Languages with Applications to Biology and Security. LNCS, vol. 9465, pp. 103–121. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25527-9_9
14. Basile, D., ter Beek, M.H.: Contract automata library. Sci. Comput. Program. **221** (2022). https://doi.org/10.1016/j.scico.2022.102841
15. Basile, D., et al.: Controller synthesis of service contracts with variability. Sci. Comput. Program. **187** (2020). https://doi.org/10.1016/j.scico.2019.102344
16. Basile, D., ter Beek, M.H., Di Giandomenico, F., Gnesi, S.: Orchestration of dynamic service product lines with featured modal contract automata. In: Proceedings of the 21st International Systems and Software Product Line Conference (SPLC 2017), vol. 2, pp. 117–122. ACM (2017). https://doi.org/10.1145/3109729.3109741
17. Basile, D., ter Beek, M.H., Legay, A.: Timed service contract automata. Innov. Syst. Softw. Eng. **2**(16), 199–214 (2020). https://doi.org/10.1007/s11334-019-00353-3
18. Basile, D., Degano, P., Ferrari, G.L.: Automata for specifying and orchestrating service contracts. Log. Meth. Comp. Sci. **12**(4:6), 1–51 (2016). https://doi.org/10.2168/LMCS-12(4:6)2016
19. Basile, D., Di Giandomenico, F., Gnesi, S., Degano, P., Ferrari, G.L.: Specifying variability in service contracts. In: Proceedings of the 11th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2017), pp. 20–27. ACM (2017). https://doi.org/10.1145/3023956.3023965
20. Basile, D., ter Beek, M.H.: A clean and efficient implementation of choreography synthesis for behavioural contracts. In: Damiani, F., Dardha, O. (eds.) COORDINATION 2021. LNCS, vol. 12717, pp. 225–238. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-78142-2_14
21. Basile, D., ter Beek, M.H.: Advancing orchestration synthesis for contract automata. J. Log. Algebr. Methods Program. (2024)
22. Basile, D., ter Beek, M.H., Pugliese, R.: Bridging the gap between supervisory control and coordination of services: synthesis of orchestrations and choreographies. In: Riis Nielson, H., Tuosto, E. (eds.) COORDINATION 2019. LNCS, vol. 11533, pp. 129–147. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22397-7_8

23. Basile, D., ter Beek, M.H., Pugliese, R.: Synthesis of orchestrations and choreographies: bridging the gap between supervisory control and coordination of services. Log. Meth. Comp. Sci. **16**(2), 9:1–9:29 (2020). https://doi.org/10.23638/LMCS-16(2:9)2020
24. Basile, D., Degano, P., Ferrari, G., Tuosto, E.: From orchestration to choreography through contract automata. In: Lanese, I., Lluch Lafuente, A., Sokolova, A., Torres Vieira, H. (eds.) Proceedings of the 7th Interaction and Concurrency Experience (ICE 2014). EPTCS, vol. 166, pp. 67–85 (2014). https://doi.org/10.4204/EPTCS.166.8
25. Basile, D., Degano, P., Ferrari, G., Tuosto, E.: Relating two automata-based models of orchestration and choreography. J. Log. Algebr. Methods Program. **85**(3), 425–446 (2016). https://doi.org/10.1016/J.JLAMP.2015.09.011
26. Basu, A., Bozga, M., Sifakis, J.: Modeling heterogeneous real-time components in BIP. In: Proceedings of the 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006), pp. 3–12. IEEE (2006). https://doi.org/10.1109/SEFM.2006.27
27. Basu, A., et al.: Rigorous component-based system design using the BIP framework. IEEE Softw. **28**(3), 41–48 (2011). https://doi.org/10.1109/MS.2011.27
28. Bauer, S.S., Mayer, P., Schroeder, A., Hennicker, R.: On weak modal compatibility, refinement, and the MIO workbench. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 175–189. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12002-2_15
29. ter Beek, M., Lenzini, G., Petrocchi, M.: Team Automata for Security Analysis of Multicast/Broadcast Communication. In: Proceedings of the ICATPN Workshop on Issues in Security and Petri Nets (WISP 2003). pp. 57–71. Eindhoven University of Technology (2003)
30. ter Beek, M.H.: Team automata—a formal approach to the modeling of collaboration between system components. Ph.D. thesis, Leiden University (2003). https://hdl.handle.net/1887/29570
31. ter Beek, M.H., Carmona, J., Hennicker, R., Kleijn, J.: Communication requirements for team automata. In: Jacquet, J.-M., Massink, M. (eds.) COORDINATION 2017. LNCS, vol. 10319, pp. 256–277. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59746-1_14
32. ter Beek, M.H., Carmona, J., Kleijn, J.: Conditions for compatibility of components. In: Margaria, T., Steffen, B. (eds.) ISoLA 2016. LNCS, vol. 9952, pp. 784–805. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47166-2_55
33. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Featured team automata. Technical report, arXiv (2021). https://doi.org/10.48550/arXiv.2108.01784
34. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Featured team automata. In: Huisman, M., Păsăreanu, C., Zhan, N. (eds.) FM 2021. LNCS, vol. 13047, pp. 483–502. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90870-6_26
35. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Can we communicate? Using dynamic logic to verify team automata (extended version). Technical report, Zenodo (2022). https://doi.org/10.5281/zenodo.7418074
36. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Can we communicate? Using dynamic logic to verify team automata. In: Chechik, M., Katoen, J.P., Leucker, M. (eds.) FM 2023. LNCS, vol. 14000, pp. 122–141. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-27481-7_9
37. ter Beek, M.H., Csuhaj-Varjú, E., Mitrana, V.: Teams of pushdown automata. Int. J. Comput. Math. **81**(2), 141–156 (2004). https://doi.org/10.1080/00207160310001650099

38. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Synchronizations in team automata for groupware systems. Technical report, TR-99-12, Leiden Institute of Advanced Computer Science, Leiden University (1999)
39. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Team automata for CSCW. In: Proceedings of the 2nd International Colloquium on Petri Net Technologies for Modelling Communication Based Systems, pp. 1–20. Fraunhofer ISST (2001)
40. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Team automata for spatial access control. In: Prinz, W., Jarke, M., Rogers, Y., Schmidt, K., Wulf, V. (eds.) Proceedings of the 7th European Conference on Computer-Supported Cooperative Work (ECSCW 2001), pp. 59–77. Kluwer (2001). https://doi.org/10.1007/0-306-48019-0_4
41. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Synchronizations in team automata for groupware systems. Comput. Sup. Coop. Work **12**(1), 21–69 (2003). https://doi.org/10.1023/A:1022407907596
42. ter Beek, M.H., Gadducci, F., Janssens, D.: A calculus for team automata. In: Ribeiro, L., Moreira, A.M. (eds.) Proceedings of the 9th Brazilian Symposium on Formal Methods (SBMF 2006), pp. 59–72. Istituto de Informatica da UFRGS, Porto Alegre (2006)
43. ter Beek, M.H., Gadducci, F., Janssens, D.: A calculus for team automata. Electron. Notes Theor. Comput. Sci. **195**, 41–55 (2008). https://doi.org/10.1016/j.entcs.2007.08.022
44. ter Beek, M.H., Hennicker, R., Kleijn, J.: Compositionality of safe communication in systems of team automata. In: Pun, V.K.I., Stolz, V., Simao, A. (eds.) ICTAC 2020. LNCS, vol. 12545, pp. 200–220. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-64276-1_11
45. ter Beek, M.H., Hennicker, R., Kleijn, J.: Compositionality of safe communication in systems of team automata. Technical report, Zenodo (2020). https://doi.org/10.5281/zenodo.4050293
46. ter Beek, M.H., Hennicker, R., Kleijn, J.: Team Automata@Work: on safe communication. In: Bliudze, S., Bocchi, L. (eds.) COORDINATION 2020. LNCS, vol. 12134, pp. 77–85. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-50029-0_5
47. ter Beek, M.H., Hennicker, R., Proença, J.: Realisability of global models of interaction. In: Ábrahám, E., Dubslaff, C., Tapia Tarifa, S.L. (eds.) ICTAC 2023. LNCS, vol. 14446, pp. 236–255. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-47963-2_15
48. ter Beek, M.H., Hennicker, R., Proença, J.: Realisability of global models of interaction (extended version). Technical report, Zenodo (2023). https://doi.org/10.5281/zenodo.8377188
49. ter Beek, M.H., Kleijn, J.: Team automata satisfying compositionality. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 381–400. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45236-2_22
50. ter Beek, M.H., Kleijn, J.: Modularity for teams of I/O automata. Inf. Process. Lett. **95**(5), 487–495 (2005). https://doi.org/10.1016/j.ipl.2005.05.012
51. ter Beek, M.H., Kleijn, J.: Associativity of infinite synchronized shuffles and team automata. Fundam. Inform. **91**(3–4), 437–461 (2009). https://doi.org/10.3233/FI-2009-0051
52. ter Beek, M.H., Kleijn, J.: Vector team automata. Theor. Comput. Sci. **429**, 21–29 (2012). https://doi.org/10.1016/j.tcs.2011.12.020

53. ter Beek, M.H., Kleijn, J.: On distributed cooperation and synchronised collaboration. J. Autom. Lang. Comb. **19**(1-4), 17–32 (2014). https://doi.org/10.25596/jalc-2014-017
54. ter Beek, M.H., Lenzini, G., Petrocchi, M.: Team automata for security – a survey. Electron. Notes Theor. Comput. Sci. **128**, 105–119 (2005). https://doi.org/10.1016/j.entcs.2004.11.044
55. ter Beek, M.H., Lenzini, G., Petrocchi, M.: A team automaton scenario for the analysis of security properties in communication protocols. J. Autom. Lang. Comb. **11**(4), 345–374 (2006). https://doi.org/10.25596/jalc-2006-345
56. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Can we Communicate? Using Dynamic Logic to Verify Team Automata (Software Artefact) (2022). https://doi.org/10.5281/zenodo.7338440. http://arcatools.org/feta
57. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30080-9_7
58. Bejleri, A., Yoshida, N.: Synchronous multiparty session types. Electr. Notes Theor. Comput. Sci. **241**, 3–33 (2008). https://doi.org/10.1016/j.entcs.2009.06.002
59. van Benthem, J., van Eijck, J., Stebletsova, V.: Modal logic, transition systems and processes. J. Log. Comput. **4**(5), 811–855 (1994). https://doi.org/10.1093/logcom/4.5.811
60. Best, E., Devillers, R.: Petri Net Primer: A Compendium on the Core Model, Analysis, and Synthesis. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-48278-6
61. Bliudze, S., van den Bos, P., Huisman, M., Rubbens, R., Safina, L.: JavaBIP meets VerCors: towards the safety of concurrent software systems in Java. In: Lambers, L., Uchitel, S. (eds.) FASE 2023. LNCS, vol. 13991, pp. 143–150. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-30826-0_8
62. Bliudze, S., et al.: Formal verification of infinite-state BIP models. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 326–343. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24953-7_25
63. Bliudze, S., Mavridou, A., Szymanek, R., Zolotukhina, A.: Exogenous coordination of concurrent software components with JavaBIP. Softw. Pract. Exper. **47**(11), 1801–1836 (2017). https://doi.org/10.1002/spe.2495
64. Bliudze, S., Sifakis, J.: The algebra of connectors: structuring interaction in BIP. IEEE Trans. Comput. **57**(10), 1315–1330 (2008). https://doi.org/10.1109/TC.2008.26
65. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are two web services compatible? In: Shan, M.-C., Dayal, U., Hsu, M. (eds.) TES 2004. LNCS, vol. 3324, pp. 15–28. Springer, Heidelberg (2005). https://doi.org/10.1007/978-3-540-31811-8_2
66. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2), 323–342 (1983). https://doi.org/10.1145/322374.322380
67. Brim, L., Cerná, I., Vareková, P., Zimmerova, B.: Component-interaction automata as a verification-oriented component-based system specification. ACM Softw. Eng. Notes **31**(2) (2006). https://doi.org/10.1145/1118537.1123063
68. Carmona, J., Kleijn, J.: Interactive behaviour of multi-component systems. In: Proceedings of the ICATPN Workshop on Token-Based Computing (ToBaCo 2004), pp. 27–31. Università di Bologna (2004)
69. Carmona, J., Kleijn, J.: Compatibility in a multi-component environment. Theor. Comput. Sci. **484**, 1–15 (2013). https://doi.org/10.1016/j.tcs.2013.03.006

70. Carmona, J., Cortadella, J.: Input/Output compatibility of reactive systems. In: Aagaard, M.D., O'Leary, J.W. (eds.) FMCAD 2002. LNCS, vol. 2517, pp. 360–377. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36126-X_22
71. Castagna, G., Gesbert, N., Padovani, L.: A theory of contracts for web services. ACM Trans. Program. Lang. Syst. **31**(5), 19:1–19:61 (2009). https://doi.org/10.1145/1538917.1538920
72. Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On global types and multi-party sessions. Log. Meth. Comp. Sci. **8**(1), 24:1–24:45 (2012). https://doi.org/10.2168/LMCS-8(1:24)2012
73. Castellani, I., Mukund, M., Thiagarajan, P.S.: Synthesizing distributed transition systems from global specifications. In: Rangan, C.P., Raman, V., Ramanujam, R. (eds.) FSTTCS 1999. LNCS, vol. 1738, pp. 219–231. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-46691-6_17
74. Clarke, D.: Coordination: Reo, nets, and logic. In: de Boer, F.S., Bonsangue, M.M., Graf, S., de Roever, W.-P. (eds.) FMCO 2007. LNCS, vol. 5382, pp. 226–256. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-92188-2_10
75. Corradini, F., Gorrieri, R., Marchignoli, D.: Towards parallelization of concurrent systems. RAIRO Theor. Informatics Appl. **32**(4–6), 99–125 (1998). https://doi.org/10.1051/ita/1998324-600991
76. Cruz, R., Proença, J.: ReoLive: analysing connectors in your browser. In: Mazzara, M., Ober, I., Salaün, G. (eds.) STAF 2018. LNCS, vol. 11176, pp. 336–350. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04771-9_25
77. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2001), pp. 109–120. ACM (2001). https://doi.org/10.1145/503209.503226
78. Dokter, K., Arbab, F.: Treo: textual syntax for Reo connectors. In: Bliudze, S., Bensalem, S. (eds.) Proceedings of the 1st International Workshop on Methods and Tools for Rigorous System Design (MeTRiD 2018). EPTCS, vol. 272, pp. 121–135 (2018). https://doi.org/10.4204/EPTCS.272.10
79. Durán, F., Ouederni, M., Salaün, G.: A generic framework for n-protocol compatibility checking. Sci. Comput. Program. **77**(7–8), 870–886 (2012). https://doi.org/10.1016/j.scico.2011.03.009
80. Edixhoven, L., Jongmans, S.S., Proença, J., Castellani, I.: Branching pomsets: design, expressiveness and applications to choreographies. J. Log. Algebr. Methods Program. **136** (2024). https://doi.org/10.1016/j.jlamp.2023.100919
81. Egidi, L., Petrocchi, M.: Modelling a secure agent with team automata. Electron. Notes Theor. Comput. Sci. **142**, 111–127 (2006). https://doi.org/10.1016/j.entcs.2004.12.046
82. Ellis, C.A.: Team automata for groupware systems. In: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge (GROUP 1997), pp. 415–424. ACM (1997). https://doi.org/10.1145/266838.267363
83. Engels, G., Groenewegen, L.: Towards team-automata-driven object-oriented collaborative work. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) Formal and Natural Computing. LNCS, vol. 2300, pp. 257–276. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45711-9_15
84. Gössler, G., Sifakis, J.: Composition for component-based modeling. Sci. Comput. Program. **55**, 161–183 (2005). https://doi.org/10.1016/j.scico.2004.05.014
85. Groote, J.F., Mousavi, M.R.: Modeling and Analysis of Communicating Systems. MIT Press, Cambridge (2014)

86. Groote, J.F., Moller, F.: Verification of parallel systems via decomposition. In: Cleaveland, R. (ed.) CONCUR 1992. LNCS, vol. 630, pp. 62–76. Springer, Heidelberg (1992). https://doi.org/10.1007/BFb0084783
87. Guanciale, R., Tuosto, E.: Realisability of pomsets. J. Log. Algebr. Methods Program. **108**, 69–89 (2019). https://doi.org/10.1016/J.JLAMP.2019.06.003
88. Harel, D., Kozen, D., Tiuryn, J.: Dynamic Logic. Foundations of Computing. MIT Press, Cambridge (2000). https://doi.org/10.7551/mitpress/2516.001.0001
89. Harel, D., Thiagarajan, P.S.: Message sequence charts. In: Lavagno, L., Martin, G., Selic, B. (eds.) UML for Real: Design of Embedded Real-Time Systems, pp. 77–105. Kluwer (2003). https://doi.org/10.1007/0-306-48738-1_4
90. Hennicker, R., Bidoit, M.: Compatibility properties of synchronously and asynchronously communicating components. Log. Meth. Comp. Sci. **14**(1), 1–31 (2018). https://doi.org/10.23638/LMCS-14(1:1)2018
91. 't Hoen, P.J., ter Beek, M.H.: A conflict-free strategy for team-based model development. In: Proceedings of the International Workshop on Process support for Distributed Team-based Software Development (PDTSD 2000), pp. 720–725. IIIS (2000)
92. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. In: POPL 2008, pp. 273–284. ACM (2008). https://doi.org/10.1145/1328438.1328472
93. International Telecommunication Union (ITU): Message Sequence Chart (MSC). Recommendation ITU-T Z.120 (2011). http://www.itu.int/rec/T-REC-Z.120
94. Jaisankar, N., Veeramalai, S., Kannan, A.: Team automata based framework for spatio-temporal RBAC model. In: Das, V.V., et al. (eds.) BAIP 2010. CCIS, vol. 70, pp. 586–591. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12214-9_106
95. Ji, Z., Wang, S., Xu, X.: Session types with multiple senders single receiver. In: Hermanns, H., Sun, J., Bu, L. (eds.) SETTA 2023. LNCS, vol. 14464, pp. 112–131. Springer, Cham (2023). https://doi.org/10.1007/978-981-99-8664-4_7
96. Jongmans, S.S.T.Q., Arbab, F.: Overview of thirty semantic formalisms for Reo. Sci. Ann. Comput. Sci. **22**(1), 201–251 (2012). https://doi.org/10.7561/SACS.2012.1.201
97. Jonsson, B.: Compositional verification of distributed systems. Ph.D. thesis, Uppsala University (1987)
98. Katoen, J.P., Lambert, L.: Pomsets for message sequence charts. In: Lahav, Y., Wolisz, A., Fischer, J., Holz, E. (eds.) Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC (SAM 1998), pp. 197–207. Humboldt-Universität zu Berlin (1998)
99. Kaynar, D.K., Lynch, N., Segala, R., Vaandrager, F.: The Theory of Timed I/O Automata, 2 edn. Synthesis Lectures on Distributed Computing Theory. Springer, Cham (2010). https://doi.org/10.1007/978-3-031-02003-2
100. Kleijn, J.: Team automata for CSCW – a survey. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems. LNCS, vol. 2472, pp. 295–320. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40022-6_15
101. Klein, J., Klüppelholz, S., Stam, A., Baier, C.: Hierarchical modeling and formal verification. an industrial case study using Reo and Vereofy. In: Salaün, G., Schätz, B. (eds.) FMICS 2011. LNCS, vol. 6959, pp. 228–243. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24431-5_17
102. Koehler, C., Clarke, D.: Decomposing port automata. In: Shin, S.Y., Ossowski, S. (eds.) Proceedings of the 24th ACM Symposium on Applied Computing (SAC 2009), pp. 1369–1373. ACM (2009). https://doi.org/10.1145/1529282.1529587

103. Kokash, N., Krause, C., de Vink, E.P.: Data-aware design and verification of service compositions with Reo and mCRL2. In: Proceedings of the 25th ACM Symposium on Applied Computing (SAC 2010), pp. 2406–2413. ACM (2010). https://doi.org/10.1145/1774088.1774590
104. Konnov, I.V., Kotek, T., Wang, Q., Veith, H., Bliudze, S., Sifakis, J.: Parameterized systems in BIP: design and model checking. In: Desharnais, J., Jagadeesan, R. (eds.) CONCUR 2016. LIPIcs, vol. 59, pp. 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPICS.CONCUR.2016.30
105. Lange, J., Tuosto, E., Yoshida, N.: From communicating machines to graphical choreographies. In: Proceedings of the 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2015), pp. 221–232. ACM (2015). https://doi.org/10.1145/2676726.2676964
106. Larsen, K.G., Nyman, U., Wąsowski, A.: Modal I/O automata for interface and product line theories. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 64–79. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71316-6_6
107. Guldstrand Larsen, K., Lorber, F., Nielsen, B.: 20 years of *real* real time model validation. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) FM 2018. LNCS, vol. 10951, pp. 22–36. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_2
108. Lauenroth, K., Pohl, K., Töhning, S.: Model checking of domain artifacts in product line engineering. In: Proceedings of the 24th International Conference on Automated Software Engineering (ASE 2009), pp. 269–280. IEEE (2009). https://doi.org/10.1109/ASE.2009.16
109. Luttik, B.: Unique parallel decomposition in branching and weak bisimulation semantics. Theor. Comput. Sci. **612**, 29–44 (2016). https://doi.org/10.1016/j.tcs.2015.10.013
110. Lynch, N.A., Tuttle, M.R.: An introduction to Input/Output automata. CWI Q. **2**(3), 219–246 (1989). https://ir.cwi.nl/pub/18164
111. Mavridou, A., Baranov, E., Bliudze, S., Sifakis, J.: Architecture diagrams: a graphical language for architecture style specification. In: Proceedings of the 9th Interaction and Concurrency Experience (ICE 2016). EPTCS, vol. 223, pp. 83–97 (2016). https://doi.org/10.4204/EPTCS.223.6
112. Milner, R., Moller, F.: Unique decomposition of processes. Theor. Comput. Sci. **107**(2), 357–363 (1993). https://doi.org/10.1016/0304-3975(93)90176-T
113. Muschevici, R., Proença, J., Clarke, D.: Modular modelling of software product lines with feature nets. In: Barthe, G., Pardo, A., Schneider, G. (eds.) SEFM 2011. LNCS, vol. 7041, pp. 318–333. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24690-6_22
114. Muschevici, R., Proença, J., Clarke, D.: Feature nets: behavioural modelling of software product lines. Softw. Sys. Model. **15**(4), 1181–1206 (2016). https://doi.org/10.1007/s10270-015-0475-z
115. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains. Part I. Theor. Comput. Sci. **13**, 85–108 (1981). https://doi.org/10.1016/0304-3975(81)90112-2
116. Oheimb, D.: Interacting state machines: a stateful approach to proving security. In: Abdallah, A.E., Ryan, P., Schneider, S. (eds.) FASec 2002. LNCS, vol. 2629, pp. 15–32. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40981-6_4

117. Orlando, S., Pasquale, V.D., Barbanera, F., Lanese, I., Tuosto, E.: Corinne, a tool for choreography automata. In: Salaün, G., Wijs, A. (eds.) FACS 2021. LNCS, vol. 13077, pp. 82–92. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90636-8_5
118. Proença, J., Madeira, A.: Taming hierarchical connectors. In: Hojjat, H., Massink, M. (eds.) FSEN 2019. LNCS, vol. 11761, pp. 186–193. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-31517-7_13
119. Proença, J.: Overview on constrained multiparty synchronisation in team automata. In: Cámara, J., Jongmans, S.S. (eds.) FACS 2023. LNCS, vol. 14485, pp. 194–205. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-52183-6_10
120. Proença, J., Clarke, D.: Typed connector families and their semantics. Sci. Comput. Program. **146**, 28–49 (2017). https://doi.org/10.1016/j.scico.2017.03.002
121. Proença, J., Clarke, D., de Vink, E., Arbab, F.: Dreams: a framework for distributed synchronous coordination. In: Proceedings of the 27th ACM Symposium on Applied Computing (SAC 2012), pp. 1510–1515. ACM (2012). https://doi.org/10.1145/2245276.2232017
122. Proença, J., Cledou, G.: ARx: reactive programming for synchronous connectors. In: Bliudze, S., Bocchi, L. (eds.) COORDINATION 2020. LNCS, vol. 12134, pp. 39–56. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-50029-0_3
123. Proença, J.: Synchronous coordination of distributed components. Ph.D. thesis, Leiden University (2011). https://hdl.handle.net/1887/17624
124. Ramadge, P.J., Wonham, W.M.: Supervisory control of a class of discrete event processes. SIAM J. Control. Optim. **25**(1), 206–230 (1987). https://doi.org/10.1137/0325013
125. Reisig, W.: Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33278-4
126. Scalas, A., Yoshida, N.: Less is more: multiparty session types revisited. Proc. ACM Program. Lang. **3**, 30:1–30:29 (2019). https://doi.org/10.1145/3290343
127. Severi, P., Dezani-Ciancaglini, M.: Observational equivalence for multiparty sessions. Fundam. Inform. **170**(1–3), 267–305 (2019). https://doi.org/10.3233/FI-2019-1863
128. Sharafi, M.: Extending team automata to evaluate software architectural design. In: Proceedings of the 32nd IEEE International Computer Software and Applications Conference (COMPSAC 2008), pp. 393–400. IEEE (2008). https://doi.org/10.1109/COMPSAC.2008.57
129. Sharafi, M., Shams Aliee, F., Movaghar, A.: A review on specifying software architectures using extended automata-based models. In: Arbab, F., Sirjani, M. (eds.) FSEN 2007. LNCS, vol. 4767, pp. 423–431. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75698-9_30
130. Smeyers, M.: A browser-based graphical editor for Reo networks. Master's thesis, Leiden University (2018). https://theses.liacs.nl/1536
131. Teren, V., Cortadella, J., Villa, T.: Decomposition of transition systems into sets of synchronizing state machines. In: Proceedings of the 24th Euromicro Conference on Digital System Design (DSD 2021), pp. 77–81. IEEE (2021). https://doi.org/10.1109/DSD53832.2021.00021
132. Teren, V., Cortadella, J., Villa, T.: Decomposition of transition systems into sets of synchronizing Free-choice Petri Nets. In: Proceedings of the 25th Euromicro Conference on Digital System Design (DSD 2022), pp. 165–173. IEEE (2022). https://doi.org/10.1109/DSD57027.2022.00031

133. Winskel, G.: An introduction to event structures. In: de Bakker, J.W., de Roever, W.P., Rozenberg, G. (eds.) REX 1988. LNCS, vol. 354, pp. 364–397. Springer, Heidelberg (1988). https://doi.org/10.1007/BFB0013026
134. Yoshida, N.: Programming language implementations with multiparty session types. In: de Boer, F.S., Damiani, F., Hähnle, R., Johnsen, E.B., Kamburjan, E. (eds.) Active Object Languages: Current Research Trends. LNCS, vol. 14360, pp. 147–165. Springer, Cham (2024). https://doi.org/10.1007/978-3-031-51060-1_6