Logical Methods in Computer Science
Volume 22, Issue 4, 2026, pp. 12:1–12:41
https://lmcs.episciences.org/

Submitted Jan. 24, 2025
Published Nov. 03, 2026

OVERVIEW AND ROADMAP OF TEAM AUTOMATA

MAURICE H. TER BEEK [a], ROLF HENNICKER [b], AND JOSÉ PROENÇA [c]

[a] CNR–ISTI, Pisa, Italy
e-mail address: maurice.terbeek@isti.cnr.it

[b] LMU Munich, Germany
e-mail address: hennicker@ifi.lmu.de

[c] CISTER and INESC TEC and University of Porto, Portugal
e-mail address: 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?), *tool support* (what has been automatised or implemented?), and *variability* (can a family of concrete product (automata) models be captured concisely?). 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.

This paper is part of the special issue *Selected Papers of the 44th International Conference on Formal Techniques for Distributed Objects, Components and Systems and the 26th International Conference on Coordination Models and Languages (FORTE and COORDINATION 2024).*

1. Introduction

Team automata (TA) were first proposed by Skip Ellis at the 1997 ACM SIGGROUP Conference on Supporting Group Work [Ell97] for modelling components of groupware systems and their interconnections. They were inspired by Input/Output (I/O) automata [LT89,LT26] and in particular inherit their distinction between internal (private, *i.e.*, not observable by other I/O automata) actions 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

Key words and phrases: Team Automata, Coordination models, Synchronous Composition, Communication properties, Realisability, Verification, Variability, Tools.

© M.H. ter Beek, R. Hennicker, and J. Proença
Creative Commons

nature, *i.e.*, input or output) as agreed upon upfront. Team automata 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 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 [tBEKR03], 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 [tBCHK17, tBHP24], which models 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 a single initial state, indicated by an incoming arrow head, typically denoted by 0, and external actions may be prefixed by "!" (for output) or "?" (for input). Further, more complex examples can also be found in the literature, increasing the comprehension of (meta) access control strategies (*e.g.*, with or without deep revocation) visualised through a virtual spaces metaphor [tBEKR01b], guaranteeing robustness against security properties (*e.g.*, integrity and secrecy) in dedicated communication protocols, or capturing communication safety in scenarios with multiple clients connecting and interacting with a server [tBCHK17, tBHK20a, tBCHP21a], possibly with a proxy for buffering [tBK09].

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.

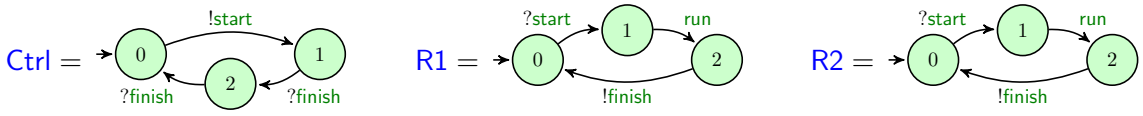


FIGURE 1. Race example: a controller Ctrl and two runners R1 and R2

Contribution This paper extends [tBHP24] in the following ways, as mentioned in detail below. In particular, we have added examples and details plus two new sections dealing with composition of systems and with variability. To help readers navigate through this paper, we include a table of contents.

Contents

1. Introduction 1
2. Team Automata in a Nutshell 5
3. Related Coordination Formalisms 8
3.1. Reo via Port Automata 8
3.2. BIP (without Priorities) 9
3.3. Contract Automata 9
3.4. Choreography Automata 10
3.5. Multi-Party Session Types 10
3.6. Other Coordination Formalisms 11
4. Communication Properties 12
4.1. Communication Requirements 13
4.2. Compliance with Communication Requirements 13
4.3. Tool Support 15
4.4. Related Work 15
4.5. Roadmap 16
5. Realisation 17
5.1. Global Models of Interaction 17
5.2. Realisation of Global Models of Interaction 18
5.3. Tool Support 20
5.4. Related Work 21
5.5. Roadmap 22
6. Composition of Systems 23
6.1. System Composition 23
6.2. Composition of STS 24
6.3. Tool Support 26
6.4. Related Work 26
6.5. Roadmap 27
7. Variability 27
7.1. Featured Component Automata and Featured Systems 29
7.2. Featured Team Automata 30
7.3. Tool Support 30
7.4. Related Work 30
7.5. Roadmap 30
8. Conclusion 31
Appendix A. Selected Publications from 25+ Years of Team Automata 33
References 34

We first revisit the specific notion of synchronisation to build team automata (Section 2), after which we briefly relate team automata to a selection of coordination models (Section 3). We then focus on four aspects of team automata that we investigated during the last five years:

verification of communication properties (Section 4), *realisation* (Section 5), *composition of systems* (Section 6, not included in [tBHP24]), and *variability* (Section 7, not included in [tBHP24]).

§4 : We report results on verification of 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 verification of communication properties.

§5 : We provide results on the decomposition (realisation) of a global interaction model[1] 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.* [CMT99] and a roadmap for future work on realisability.

§6 : We discuss results on the composition of *systems* of component automata and the composition of synchronisation type specifications, as well as on the preservation of communication properties from team automata of subsystems to global team automata. We provide a roadmap for future work on composition of systems.

§7 : We present results on team automata to support variability in the development and analysis of teams, *i.e.*, featured team automata that concisely describe a family of concrete product (automaton) models for specific configurations determined by feature selection, show how to lift the notion of receptiveness to the level of family models, and mention a roadmap for future work on variability.

Finally, we mention further aspects of team automata and of some of the related coordination models and conclude the paper (Section 8). Figure 2 summarises this paper's contribution. Sect. A lists selected team automata publications.

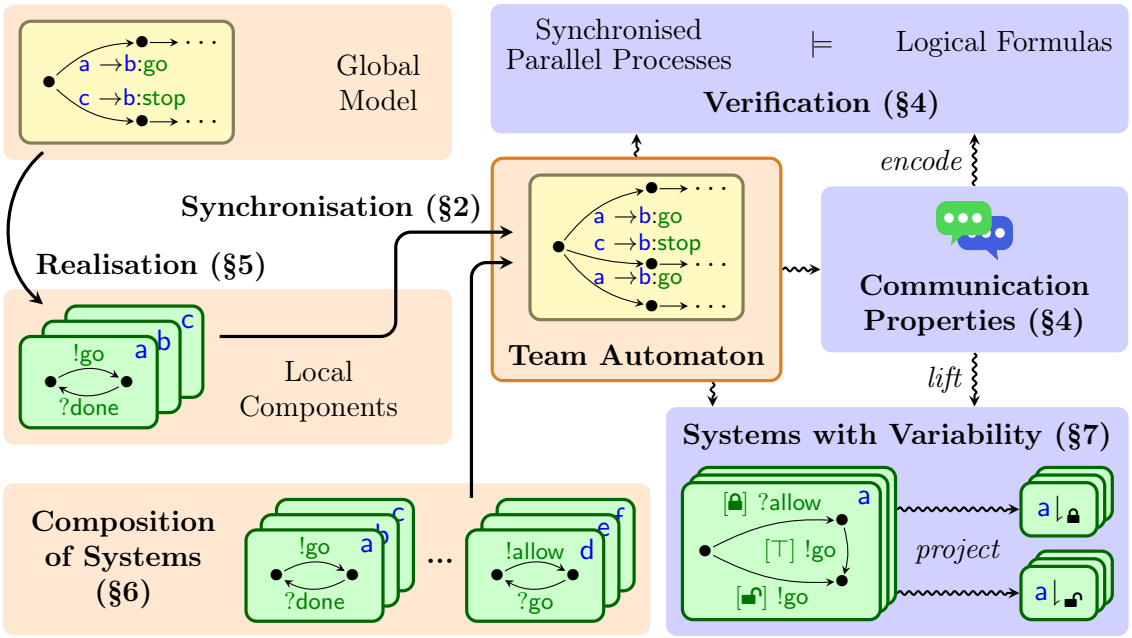


FIGURE 2. Aspects of team automata addressed in this paper

[1]This global model is explicitly specified upfront, while the team automaton results from the composition of component automata (*cf.* Sect. 5)

2. Team Automata in a Nutshell

Team automata were originally introduced by Ellis [Ell97] and formally defined in [tBEKR03]. 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 quadruple $\mathcal{L} = (Q, q_0, \Sigma, E)$ such that Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite set of 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 $a_1, \ldots, a_n \in \Gamma$. For $\Gamma \subseteq \Sigma$ and $a \in \Sigma$, we write $q \xrightarrow{\Gamma^*; a} q''$ if there exists q' such that $q \xrightarrow{\Gamma}{}^* q'$ and $q' \xrightarrow{a} q''$. 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 $\mathcal{R}(\mathcal{L})$.

Component Automata. A *component automaton* (CA) is an LTS $\mathcal{A} = (Q, q_0, \Sigma, E)$ such that $\Sigma = \Sigma^? \cup \Sigma^! \cup \Sigma^\tau$ is a set of *component actions* (or simply *actions*) 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 $\mathsf{lts}(\mathcal{S}) = (Q, q_0, \Lambda(\mathcal{S}), E(\mathcal{S}))$, 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 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 component actions as an input action and in (at least) one set Σ_ℓ of action labels as an output action. System actions that are neither communicating nor internal to a component are called *open*. Hence, the set of open system actions is given by $\Sigma^\circ = \Sigma \setminus (\Sigma^\bullet \cup \bigcup_{n\in\mathcal{N}} \Sigma_n^\tau)$. The system is *closed* if $\Sigma^\circ = \varnothing$, *i.e.*, all non-communicating actions are internal component actions.

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. Obviously, if $a \in \Sigma^\circ$ is an open system action, then either out or in must be empty. 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:

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

Note that $\Lambda(\mathcal{S})$ depends only on $\mathcal{N}$ and the sets Σ_n of component actions for each $n \in \mathcal{N}$. If $\mathsf{out} = \{n\}$ is a singleton, we write (n, a, in) instead of $(\{n\}, a, \mathsf{in})$; similarly for singleton sets in.

In all figures and examples, interactions $(\mathsf{out}, a, \mathsf{in})$ are presented by the notation $\mathsf{out} \to \mathsf{in} : \mathsf{a}$ 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

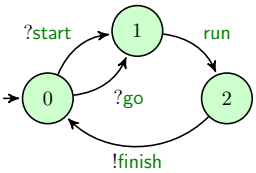
- either $\lambda = (\mathsf{out}, a, \mathsf{in})$ and:
  $q_n \xrightarrow{a}_{\mathcal{A}_n} q'_n$, for all $n \in \mathsf{out} \cup \mathsf{in}$, and $q_m = q'_m$, for all $m \in \mathcal{N} \backslash (\mathsf{out} \cup \mathsf{in})$;
- or $\lambda = (n, a)$, $a \in \Sigma^\tau_n$ is an internal 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 2.1. The Race system in Fig. 1 is a closed system. It 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 Section 1, 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.

As an open system, consider a variant of the Race example from Fig. 1 in which runner R2 is replaced by R2′ depicted on the right.[2] The difference is that this runner may decide not to wait for the start signal of the controller and start by performing an input action go. This is an external action but not a communicating action, which may be called by the system's environment (*e.g.*, a false start signal coming from the outside). Thus, this Race system has an open input. As a result, the undesired state $(0,0,1)$ becomes reachable with the external go action. ▷

R2′ =



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 $\mathsf{st} : \Sigma^\bullet \to \mathsf{Intv} \times \mathsf{Intv}$ that assigns to any communicating action a an individual synchronisation type $\mathsf{st}(a)$. A system label $\lambda = (\mathsf{out}, a, \mathsf{in})$ *satisfies* $\mathsf{st}(a) = (O, I)$, denoted by $\lambda \models \mathsf{st}(a)$, if $|\mathsf{out}| \in O \wedge |\mathsf{in}| \in I$.

[2] Another variant of this Race system as an open system will be described in Section 6 (*cf.* Example 6.1).

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.

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

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 and open actions no restriction is applied. The reason is that (1) an internal action of a component can always be executed when it is locally enabled and (2) an open action is meant to be restricted only when composing systems (*cf.* Section 6).

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

$$\mathbf{ta}(\mathcal{S},\mathbf{st}) = (Q, q_0, \Lambda(\mathcal{S},\mathbf{st}), E(\mathcal{S},\mathbf{st})).^{3}$$

We write $\Lambda(\mathbf{st})$ and $E(\mathbf{st})$ 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 $\Lambda(\mathbf{st}) \neq \varnothing$. Labels in $\Lambda(\mathbf{st})$ are called *team labels* and transitions in $E(\mathbf{st})$ are called *team transitions*.

Example 2.2. 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 $\mathsf{n} : \mathsf{a}$. ▷

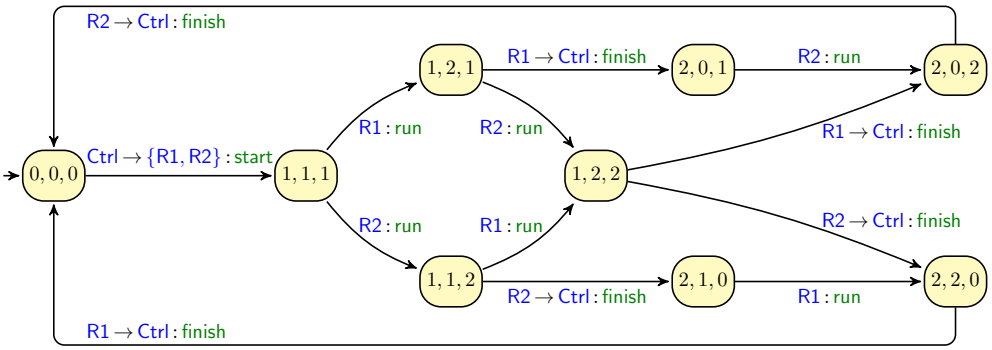


FIGURE 3. Team automaton of the Race system example in Fig. 1

Example 2.2 uses only STS with intervals of size one, as in all the examples in this paper. Further, more complex examples of STS that exploit larger intervals can be found in the literature (*cf.*, *e.g.*, [tBCHK17, tBHK20a]). We now recall some familiar ones from [tBCHK17].[4]

[3]Starting with [tBHK20a], we have been using 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 [Ell97, tBEKR03] and subsequent papers, where actions $a \in \Sigma$ have been used in team transitions. However, to study communication properties [tBCHP23], compositionality [tBHK20a] and realisability [tBHP23a], explicit rendering of the CA that actually participate in a transition of the team turned out useful.

[4]The abbreviation '*ai*' was introduced in [tBEKR03] and stands for *action indispensable*, meaning that all CA that have the action in their set Σ of actions have to participate in the synchronisation.

([1,1], [0,∗]): multicast communication, meaning that a communicating action can be executed only as a synchronisation involving exactly one component for which it is an output action and any number of the components in which it is an input action. This is called weak synchronisation in BIP.

([1,1], *ai*): (strong) broadcast communication, meaning that whenever a communicating action is executed it occurs exactly once in its output role in that transition with all input components involved. This is called strong synchronisation in BIP.

(*ai*, *ai*): transitions on communicating actions are always 'full' synchronisations, meaning that all components that share a communicating action are involved in all transitions on that action. In case all external actions are communicating (*i.e.*, in a closed system), this means that we are dealing with the classical synchronous product of automata (*cf.*, *e.g.*, [Arn94, CK13, tBCK16]).

([1,∗], [0,∗]): transitions on communicating actions always involve at least one component where that action is an output action. This is the idea of 'master-slave' communication (*cf.* [tBEKR03]), according to which a master (output) can always be executed and slaves (input) never proceed on their own.

([1,∗], [1,∗]): as directly above, but now at least one slave has to 'obey' (the master). This is called 'strong master-slave' communication (*cf.* [tBEKR03]), by which a master (output) can always be executed and slaves (input) must be involved.

3. Related Coordination Formalisms

In this section, we briefly relate team automata to a selection of formal coordination models and languages. This selection is based on a more detailed comparison in [tBHP24], which provides for each formalism, (1) the definition of the variant considered, (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.

3.1. Reo via Port Automata.

Reo [Arb04, JA12] is a coordination language to specify and compose *connectors*, *i.e.*, patterns of valid synchronous interactions of ports of components or other connectors. Constraint automata [BSAR06] is a reference model for Reo's semantics [JA12]. Many variants of constraint automata exist [JA12, Sect. 3.2.2], some distinguishing inputs from outputs as in team automata. A variant called *port automata* [KC09] abstracts away from *data* constraints, focusing on *synchronisation* (of actions) and *composition* (of automata). Synchronisation types in team automata restrict the number of inputs and outputs of ports with shared names; synchronisation in port automata force how this is done. No variant uses a similar notion to team automata's synchronisation types, although they can be expressed using intermediate ports.

There are tools to *analyse*, *edit*, *visualise*, and *execute* Reo connectors. Analyses include model checking, using either the dedicated model checker Vereofy [KKSB11] or encoding Reo into mCRL2 [KKdV10, PM19], and simulation of extensions with parameters [PC17] and with reactive programming notions [PC20], many accessible online at `http://arcatools.org/reo`. Editors and visualisation engines include an Eclipse-based implementation [AKM+08b] and editors based on JavaScript that run in a browser [Sme18, CP18]. Execution engines for Reo include a Java-based implementation [DA18] and a distributed engine using actors in Scala [Pro11, PCdVA12].

3.2. **BIP (without Priorities).** BIP [BBS06, BS08] 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 [BS08]. We have previously compared team automata with BIP in [tBCHK17], describing how some explicit patterns of interaction of BIP, such as broadcasts, are modelled in team automata (*cf.* Section 2), and with BIP without priorities [KKW+16] in [tBHP24], presenting a formalisation of BIP without priorities in a style similar to team automata that facilitated the comparison. There are some technical core differences between these formalisations.

BIP's formalisation does not include explicit internal actions, although they do exist at implementation level (*e.g.*, in JavaBIP [BMSZ17]). BIP's *synchronisation* mechanism ignores inputs and outputs. However, the flow of *data* at each interaction is sometimes described orthogonally [BMSZ17], 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 [BB20]. These differences reflect a different focus: less emphasis on *communication properties* (Section 4), internal behaviour of local components, and *realisability* notions (Section 5); yet more focus on the exploration of different formalisms for the *composition* of interactions and programs, supported by tools to connect to running systems [BMSZ17]. Similar to team automata's synchronisation types, BIP has a formalisation parameterised on the number of components [MBBS16]. 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.

Several tools exist for BIP, including verification tools that traverse the state space of BIP programs [BCJ+15] or use the VerCors model checker [BvdBH+23], and a toolset LALT-BIP for verifying freedom from global and local deadlocks [ABB+18]. BIP also has a C++ reference engine [BBB+11] and a Java engine called JavaBIP [BMSZ17].

3.3. **Contract Automata.** Contract automata [BDFT14,BDF16] 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 [BDGG+17, BtBD+20] 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 [BtBL20]. *Composition* of contract automata is a variant of the synchronous (automata) 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 (sychronisation) must happen. A composition is in *agreement* if each request is matched with an offer. Contract automata with committed states as introduced in [BtB24] can mimick multi-party *synchronisation* (by forcing two concatenated transitions with an intermediate committed state to be executed atomically). Internal actions can be modelled as offer actions (which do not interfere with agreement) or as silent (τ) actions as introduced in [BtB24].

In [BDFT16], contract automata are compared with communicating machines [BZ83]. To guarantee that a composition of contract automata corresponds to a well-behaving (*i.e.*,

realisable, *cf.* Section 5) choreography,[5] 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 [BtBD+20], this condition is related to the phenomenon of *state sharing* in team automata [EG02], 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 [CK04,tBK12]. The relation between the branching condition of contract automata and (non-)state-sharing in (vector) team automata needs further study.

Contract automata are supported by a software API called Contract Automata Library (CATLib) [BtB22], which a developer can exploit to specify contract automata and perform operations like composition and synthesis. The synthesis operation uses supervisory control theory [RW87], properly revisited in [BtBP20] 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 [BDF16,BtBD+20,BtBP20]. CATLib was designed to be easily extendable to support related formalisms; it currently supports synchronous communicating machines [LTY15].

3.4. **Choreography Automata.** Choreography automata [BLT20] are automata with labels that describe interactions (sender, receiver, and message name), very similar to contract automata. Internal actions are not captured by choreography automata (as in Reo and BIP's formalisations) and only binary *synchronisation* (of actions) is supported (as in contract automata): each such interaction has a single sender (the agent that *offers*) and a single receiver (the agent that *requests*). Desirable properties of choreography automata include deadlock-freedom, among others, focused on the language accepted by these automata [BLT22]. Consequently, properties that rely on observational equivalence notions such as bisimilarity are not covered by these analyses.

Corinne [ODPB+21] can be used to visualise choreography automata and to automatise operations like composition (to combine local automata into a single global automaton representing combined behaviour, *e.g.*, through synchronisation of actions), projection (to derive local automata from a global composite automaton), and checking for well-formedness (conditions to ensure communication properties like deadlock-freedom, *cf.* Section 4).[6]

3.5. **Multi-Party Session Types.** Multi-party session types [HYC08, BY09, SY19] are a well-established family of typing disciplines defined on top of session calculi to describe communication protocols involving multiple session participants (*multi-party*). Some variants of multi-party session types are asynchronous [HYC08], in which the communication channels on which the parties interact are buffered, allowing participants to read or write messages (which may carry *data*). Other variants are synchronous [BY09, SY19], more similar to

[5] A choreography can be seen as a global viewpoint specifying the interactions among distributed participants in terms of a *contract* defining their expected communication behaviour in terms of message exchanges. Such a choreography is well-behaving if it is realisable by local behaviour conforming to the global viewpoint.

[6] As admitted by the authors themselves in [BLT23], the well-formedness condition for choreography automata reported in [BLT20, Def. 4.13] was flawed, invalidating [BLT20, Thm 4.15] on communication properties (liveness, lock freedom, and deadlock freedom) of choreography automata. In [BLT23], the authors show that choreography automata can be conceived as specifications of formal choreographic languages, which can be used to prove the communication properties of a given choreography automaton.

team automata, in which communication is entirely synchronous without the use of buffered channels. Some of the early asynchronous variants (like the ones used by Honda *et al.* [HYC08]) and some of the early synchronous variants (like [BY09]) support some form of multicasting to multiple (explicit) receivers.

Most approaches to multi-party session types distinguish (1) a *global type*, often a starting point, describing the whole communication protocol of the *composed* system; and (2) the *local types*, often derived from the global type, describing the contributions of the individual participants to the protocol. Multi-party session types typically require *projectability*, *i.e.*, a global type can be projected onto consistent local types for each participant, a property closely related to realisability (*cf.* Section 5). Synchronisation types in team automata, on the other hand, are essentially *capacity types for communicating actions, which specify a range for the numbers of senders and a range for the number of receivers that can participate in such actions.* A special case is unicast communication. In contrast to team automata, neither global nor local types support internal actions, since their focus is on inter-process communication within a session, thus deliberately ignoring the internal behaviour of processes. Team automata, however, do not support communication of data, which is an important ingredient of the typing discipline of session types.

Synchronous multi-party session types support a limited set of communication patterns when compared to team automata (*cf.* Section 2). Multi-party session types allow useful behavioural properties to be statically verified, like the absence of communication errors and, in case of simple session types without session interleaving, also deadlock-freedom (*e.g.*, in the synchronous multi-party session types by Dezani-Ciancaglini *et al.* [DGJ$^+$15, SD19]). To ensure deadlock-freedom in general multi-party session types, additional type systems (like the interaction typing proposed in [BCD$^+$08, CDYP16]) are required to keep track of causality across sessions and forbid causality cycles in the overall behaviour.

There exists a rich variety of tools on variants of multi-party session types, as reported by Yoshida in a recent survey [Yos24] of integrations of multi-party session types into different programming languages. This survey reflects a core focus in session types theory on increasing trustworthiness of implementations of distributed systems by statically guaranteeing properties like deadlock-freedom.

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 [BDL04, LLN18] and mCRL2 [GM14, AG23], as well as more generic formalisms like message sequence charts [HT03, ITU11], event structures [NPW81, Win88], Petri nets [Rei13, BD24], and I/O automata [LT89, KLSV10, LT26]. Without pretending completeness, we discuss some of these in this section.

Uppaal. 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 team automata. 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. 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 processes (components), yet quite verbose, which we exploited to verify communication properties of team automata [tBCHP23].

Message Sequence Charts. 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. Katoen and Lambert [KL98] have used *pomsets* to formalise message sequence charts, 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 [GT19], and Edixhoven *et al.* extended pomsets with a hierarchical structure [EJPC24], reasoned over realisability, and compared it to event structures [NPW81] (often used to give semantics to Petri nets).

Petri Nets. 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 [tBK12], a subclass of team automata—non-state-sharing vector team automata—has been encoded into *Individual Token Net Controllers*—a model of vector-labeled Petri nets—covering team automata in which the synchronisation of a set of agents cannot be influenced by the remaining agents (*cf.* Section 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 [Cla07]. This extension could also be used to model the synchronisation mechanism of team automata, although the analysis of these nets is non-trivial.

I/O Automata. I/O automata and related formalisms like I/O systems [Jon87], interface automata [dAH01], reactive transition systems [CC02], interacting state machines [vO02], and component-interaction (CI) automata [BČVZ06] all distinguish input, output, and internal actions. However, I/O automata are, by definition, *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 [tBK05]. All formalisms define composition as the synchronous product of automata except for interface automata [dAH01], 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 [tBHK20a] (*cf.* Sect. 2).

4. Communication Properties

Compatibility of components is an important issue for systems to guarantee successful (*safe*) communication [BSBM05, CGP09, DOS12, CK13, BCZ15, tBCK16, tBCHK17, tBHK20c], *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 [tBCHK17], 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; in [tBCHK17] it was said to be *weakly compliant* if the communication can eventually occur after some internal actions have been performed (akin to weak compatibility [BMSH10, HB18] or agreement of lazy request actions [BtBD$^+$20]). This idea was generalised in [tBHK20c, tBHK20a] where *arbitrary* actions, not only internal ones, are allowed before the required communication happens. In this paper, we use this more general notion of weak compliance. 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 [tBHK20c, tBHK20a].

4.1. **Communication Requirements.** The assignment of a single synchronisation type to a team automaton 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 [tBHK20a], *synchronisation type specifications* which assign a synchronisation type individually to each communicating action. As we have seen in Section 2, such specifications uniquely determine a team automaton. Any synchronisation type specification generates communication requirements to be satisfied by the team.

Receptiveness. If, in a reachable state q of $\mathsf{ta}(\mathcal{S}, \mathsf{st})$, 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 a communicating output action $a \in \Sigma^\bullet$, *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 $\mathsf{st}(a) = (O, I)$, *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 $\mathsf{rcp}(\mathsf{out}, a)@q$. If $\mathsf{out} = \{n\}$, we write $\mathsf{rcp}(n, a)@q$ for $\mathsf{rcp}(\{n\}, a)@q$.

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

4.2. **Compliance with Communication Requirements.** The TA $\mathsf{ta}(\mathcal{S}, \mathsf{st})$ is compliant with a receptiveness requirement $\mathsf{rcp}(\mathsf{out}, a)@q$ 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 like

in [tBCHK17]) before accepting a, then we speak of weak receptiveness [tBHK20a]. 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 $\exists_{\mathsf{in}}\,.\,q \xrightarrow{(\mathsf{out},a,\mathsf{in})}_{\mathbf{ta}(\mathcal{S},\mathbf{st})}$ while it is *weakly compliant* with $\mathbf{rcp}(\mathsf{out},a)@q$ if $\exists_{\mathsf{in}}\,.\,q \xrightarrow{(\Lambda(\mathbf{st})_{\setminus\mathsf{out}})^*\,;\,(\mathsf{out},a,\mathsf{in})}_{\mathbf{ta}(\mathcal{S},\mathbf{st})}$, where $\Lambda(\mathbf{st})_{\setminus\mathsf{out}}$ denotes the set of team labels in which no component of out participates. Formally, $\Lambda(\mathbf{st})_{\setminus\mathsf{out}} = \{\,(\mathsf{out}',a,\mathsf{in}) \in \Lambda(\mathbf{st}) \mid (\mathsf{out}' \cup \mathsf{in}) \cap \mathsf{out} = \varnothing\,\} \cup \{\,(n,a) \in \Lambda(\mathbf{st}) \mid n \notin \mathsf{out}\,\}$.

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

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 4.1 (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.[7]

Example 4.2 (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 with these responsiveness requirements. There are four more responsiveness requirements when the controller is in state 2, two of which are only weakly fulfilled. ▷

As far as we know, such powerful compliance notions for I/O-based, synchronous component systems were not studied before, which we will explain better in Sect. 4.4. For one, I/O

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

automata are, by definition, input-enabled. This means that, differently from component automata and team automata, in any state any input action will be accepted. Hence any I/O automaton is receptive.

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 [tBHK20a, tBHK20b, tBCHP21b].

Example 4.3 (Compliance of Open Systems)**.** Consider the open system variant Race′ of the Race example from Example 2.1. Due to the open input action, state $(0,0,1)$ is reachable and has a receptiveness requirement of the controller who wants to start the competition, expressed by $\mathsf{rcp}(\mathsf{Ctrl},\mathsf{start})@(0,0,1)$. Clearly, the TA $\mathsf{ta}(\mathsf{Race}',\mathsf{st}_{\mathsf{Race}})$ is *not* (weakly) compliant with this receptiveness requirement at $(0,0,1)$, since the second runner would also be needed to start. So we must choose a different synchronisation policy. The solution is simple. We just omit transition $(0,0,0) \xrightarrow{(\varnothing,\mathsf{go},\mathsf{R2}')} (0,0,1)$. Then $(0,0,1)$ is no longer reachable and the new team is compliant with the receptiveness requirements. Removal of the 'bad' open input transition matches the approach of interface automata in [dAH01], where components are considered to be *compatible* if they can work properly together in a 'helpful' environment. ▷

4.3. **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 [HKT00] using (complex) interactions as modalities (*cf.* [tBCHP22a, tBCHP23]). 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 [tBCHP22b] to support our theory for closed systems. It implements a transformation of CA, systems, and TA into mCRL2 [AG23] 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.

4.4. **Related Work.** The genericity of our approach with respect to 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]), meaning that a communicating action can be executed only as a synchronisation involving exactly one component for which it is an output action and exactly one for which it is an input action (*cf.* our discussion in [tBCHK17] and in Section 2). Our notion of receptiveness is inspired

by the compatibility notion of *interface automata* [dAH01] and indeed both notions coincide for closed systems and 1-to-1 communication. It also coincides with receptiveness in [CC02].

Weak receptiveness is inspired by the notion of weak compatibility in [BMSH10] and also corresponds to unspecified reception in the context of n-protocol compatibility in [DOS12] and lazy request in *contract automata* [BtBD$^+$20]. We are not aware of compatibility notions concerning responsiveness. In [CC02], it is captured by deadlock-freedom and in [DOS12] 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 inspired by de Alfaro and Henzinger [dAH05] is subtle. Note that the distinction between our input and output actions is not relevant for the more traditional notions of deadlock and that two types of deadlocks are often distinguished: global and local (*cf.*, *e.g.*, [ABB$^+$18]). In short, global deadlock is a property of the whole system/composition (*i.e.*, no further interaction is enabled), whereas local deadlock is a liveness failure of a component or subsystem within that system/composition (*i.e.*, some component(s) can no longer progress while the system/composition as a whole can). Clearly, global deadlock implies local deadlock but not vice versa, which illustrates the importance of studying local deadlocks: proving only global deadlock-freedom may ignore situations in which part of the system is permanently deadlocked, while the rest can continue to execute.

Weak receptiveness and weak responsiveness can be used to guarantee (global) deadlock-freedom, in the sense of BIP [GS05], choreography automata [BLT20], or stuck-freeness in multi-party session types [SY19]. This requires some assumptions (*e.g.*, on the synchronisation types) and we refer to [tBHK20c] for a more detailed discussion. Global deadlock-freedom does, however, not imply (weak) receptiveness. Consider, for instance, a system state where Alice can send either a message a or a message b to Bob, but Bob can only receive a. This state is deadlock-free but it is not receptive, since Alice can never send b. Similarly, global deadlock-freedom does not imply (weak) responsiveness. Consider, for instance, a system state where Alice can receive a forever, Bob can receive b forever, and Carol can send a forever. If no other participants exist, then this state is deadlock-free but it is not responsive, since Bob can never receive b.

These examples point out the crucial difference between global and local deadlocks, covered by the notions of "individual deadlock" in BIP [GS05], "lock" in choreography automata [BLT20], and "strong lock" in multi-party session types [SY19]. The difference between the latter two is that [BLT20] assumes fair runs. The notion of individual deadlock in [GS05] is defined differently, but seems equivalent to a "lock" in [BLT20] 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. Hence, this is also equivalent to lock-freedom in [BLT20]. Strong lock-freedom in multi-party session types [SY19] 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.*, [AKM08a, KKSB11, KKdV10, PM19]).

4.5. **Roadmap.** In [tBHK20c], 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. To make this distinction between possible and required communication explicit,

we proposed to introduce designated states in CA, where execution can stop but may also continue, next to states where progress is required.[8] The introduction of such *safe* 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.

5. Realisation

In this section, we consider a top-down method by which first a global model $\mathcal{M}$ for the intended interaction behaviour of a system is provided (on the basis of a system signature and synchronisation type specification $\mathbf{st}$). Then we look for a system $\mathcal{S}$ of component automata, to be used as a *realisation of* $\mathcal{M}$, whose generated team automaton $\mathbf{ta}(\mathcal{S},\mathbf{st})$ is bisimilar to the global model $\mathcal{M}$. Our approach is restricted to the case in which components do not contain internal actions. Hence bisimilarity is strong.

For our realisation method we instantiate the generic approach investigated in [tBHP23a] by applying 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 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 [tBHP23a] as well, but they are not relevant for team automata.

Recall from Section 2 the notions of component automaton (CA), system $\mathcal{S}$, synchronisation type specification $\mathbf{st}$, and generated team automaton (TA) $\mathbf{ta}(\mathcal{S},\mathbf{st})$. Our realisation method is summarised in Fig. 4 and will be explained in more detail in the next sections.

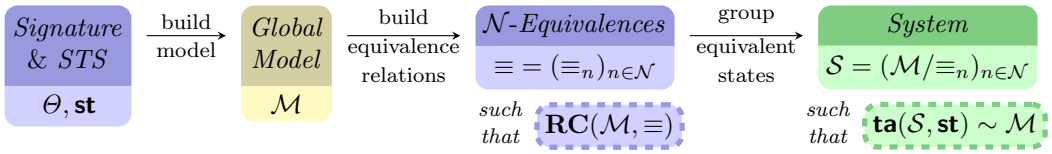


Figure 4. Realisation method

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 $\Sigma_n = \Sigma_n^? \cup \Sigma_n^!$ split into disjoint sets $\Sigma_n^?$ of *input actions* and $\Sigma_n^!$ of *output actions*. As in Section 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 Section 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 $\mathcal{M} = (Q,q_0,\Lambda(\Theta,\mathbf{st}),E)$.

[8] In [tBHK20c], we called these final states, but we now prefer to call them *safe states*. Roughly speaking, these are states where automata can break a communication requirement without raising a communication error. A typical example would be when a server component is ready to receive an input which never arrives.

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

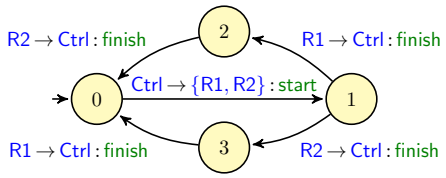


FIGURE 5. Interaction set Λ_{Race} and global interaction model $\mathcal{M}_{\mathsf{Race}}$

Example 5.1. 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 2.2, we use the STS $\mathbf{st}_{\mathsf{Race}}$ with $\mathbf{st}_{\mathsf{Race}}(\mathsf{start}) = ([1,1],[2,2])$ and $\mathbf{st}_{\mathsf{Race}}(\mathsf{finish}) = ([1,1],[1,1])$.

Fig. 5 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. ▷

Remark 5.2. Our development methodology can be extended by providing first an abstract specification of desired and forbidden interaction properties from a global perspective. In [tBHP23a], 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."*

$$\Big[\mathit{some}^*; \mathsf{Ctrl} \to \{\mathsf{R1}, \mathsf{R2}\} : \mathsf{start}\Big] \left(\begin{array}{l} \langle \mathit{some}^*; \mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}\rangle\, \mathit{true} \wedge \\ \langle \mathit{some}^*; \mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish}\rangle\, \mathit{true} \end{array} \right)$$

(2) *"No runner should finish before she/he was started by the controller."*

$$\left[\Big(- \left(\mathsf{Ctrl} \to \{\mathsf{R1}, \mathsf{R2}\} : \mathsf{start}\right) \Big)^* \; ; \; \left(\begin{array}{l} \mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish} \; + \\ \mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish} \end{array} \right) \right] \mathit{false}$$

To check that a global interaction model satisfies a specification, we propose to use the mCRL2 toolset [GM14, AG23]. For this purpose, as explained in [tBCHP23], 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 $\mathcal{M}$ with respect to $\mathbf{st}$ if the team automaton $\mathbf{ta}(\mathcal{S}, \mathbf{st})$ generated over $\mathcal{S}$ by $\mathbf{st}$ (as defined in Section 2) is bisimilar to $\mathcal{M}$, *i.e.*, $\mathbf{ta}(\mathcal{S}, \mathbf{st}) \sim \mathcal{M}$. 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 5.3. Technically, the definition of realisability in [tBHP23a] uses a synchronous Γ-composition $\otimes_\Gamma(\mathcal{A}_n)_{n\in\mathcal{N}}$ [tBHP23a, 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 TA $\mathbf{ta}(\mathcal{S}, \mathbf{st})$. 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.* [vBvES94]), 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 [tBCHP23, tBHP23a].

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 n. 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 $n \in \mathcal{N}$ 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 sufficient 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.

$\mathbf{RC}(\mathcal{M}, \equiv)$: 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 all global transitions $q_n \xrightarrow{(\mathsf{out}_n,a,\mathsf{in}_n)}_{\mathcal{M}} q'_n$ such that $n \in (\mathsf{out}_n \cap \mathsf{out}) \cup (\mathsf{in}_n \cap \mathsf{in})$, there is a global transition $g \xrightarrow{(\mathsf{out},a,\mathsf{in})}_{\mathcal{M}} g'$ such that $q'_n \equiv_n g'$ for each $n \in \mathsf{out} \cup \mathsf{in}$.

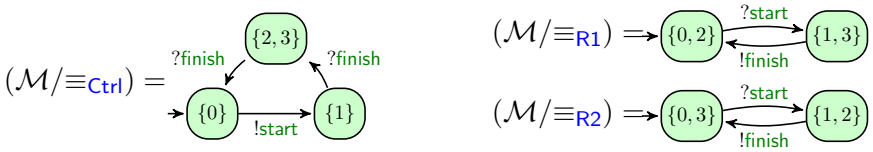
The intuition for this requirement is that if component n can participate in executing an output (input, resp.) action a in state q_n, then n should also be able to participate in executing output (input, resp.) action a in state g, since n cannot distinguish q_n and g. As this should hold for all $n \in \mathsf{out} \cup \mathsf{in}$, 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 [tBHP23a], 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

- $Q/{\equiv_n} = \{\, [q]_{\equiv_n} \mid q \in Q \,\}$ and
- $E/{\equiv_n}$ is the least set of transitions generated by rule $\dfrac{q \xrightarrow{(\mathsf{out},a,\mathsf{in})}_{\mathcal{M}} q' \qquad n \in \mathsf{out} \cup \mathsf{in}}{[q]_{\equiv_n} \xrightarrow{\mathsf{a}}_{(\mathcal{M}/{\equiv_n})} [q']_{\equiv_n}}$

Example 5.4. Take the global LTS $\mathcal{M}_{\mathsf{Race}}$ in Fig. 5. 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:



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

5.3. **Tool Support.** We implemented a prototypical tool, called Ceta, to perform realisability checks and system synthesis (*cf.* [tBHP23a, tBHP23b]). 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 with respect to 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.

5.4. **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.* [CMT99], followed by a brief comparison with other approaches.

Relationship to [CMT99]. 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 [CMT99, Theorem 3.1]. The main differences are:

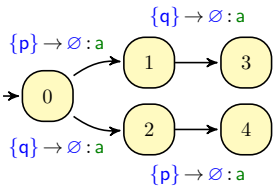
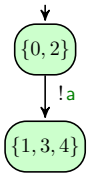
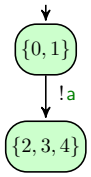
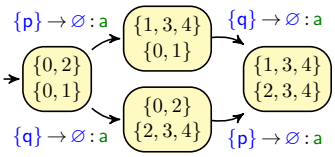
(1) In [CMT99], there is no distinction between input and output actions.
(2) In [CMT99], interactions are always full synchronisations on an action a, while we deal with individual multi-interactions $(\mathsf{out}, a, \mathsf{in})$ specified by an STS. Of course, we can also use an STS $\mathbf{st}_{full}$ 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 [CMT99], 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 [CMT99, 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 Example 5.5 below 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. In fact, Example 5.5 shows that bisimulation allows the merging of of states which would be distinct under isomporphism. Note that [CMT99, 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.

Example 5.5. There is a basic example for a situation, depicted in Table 1, in which a global LTS $\mathcal{M}$ satisfies our realisability condition but there is no realisation that is isomorphic to $\mathcal{M}$ (only bisimilar). In this example, let $\Sigma = (I, M)$, with $I = \{\mathsf{p}, \mathsf{q}\}$ and $M = \{\mathsf{a}\}$, and let $\Gamma = \{\mathsf{p} \to \varnothing : \mathsf{a}, \mathsf{q} \to \varnothing : \mathsf{a}\}$ be the interaction set. The global LTS $\mathcal{M}$ is depicted on the left side of Table 1 with five states and four transitions. As equivalences for p and q we take the reflexive, symmetric, and transitive closures of $1 \equiv_\mathsf{p} 3$, $0 \equiv_\mathsf{p} 2$, and $1 \equiv_\mathsf{p} 4$, and of $0 \equiv_\mathsf{q} 1$, $2 \equiv_\mathsf{q} 4$, and $2 \equiv_\mathsf{q} 3$. Note that $1 \equiv_\mathsf{p} 4$ and $2 \equiv_\mathsf{q} 3$ are enforced to satisfy $RC(\mathcal{M}, \equiv)^\mathsf{r}$. By symmetry and transitivity, we also get $3 \equiv_\mathsf{p} 4$ and $3 \equiv_\mathsf{q} 4$. Since the realisability condition holds, $\mathcal{M}$ is realisable with rich local actions up to bisimilarity. The local components $\mathcal{M}_\mathsf{p}$ and $\mathcal{M}_\mathsf{q}$ are depicted in the middle of Table 1. However, $\mathcal{M}$ is not realisable up to isomorphism. Otherwise, states 3 and 4 should be the same (*cf.* [CMT99, Theorem 3.1]). ▷

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 [BLT22, CDP12] with realisability expressed by language equivalence.

Table 1. A global model (left) [tBHP23b, Example 8] that is realisable for bisimilarity but not for isomorphism; the realisation (middle); and the re-composed TA with $\mathcal{S} = (\{p, q\}, \{\mathcal{M}_p, \mathcal{M}_q\})$ and $\mathbf{st}(\mathsf{a}) = ([1, 1], [0, 0])$

Global $\mathcal{M}$	Local $\mathcal{M}_\mathsf{p}$	Local $\mathcal{M}_\mathsf{q}$	$\mathbf{ta}(\mathcal{S}, \mathbf{st})$
			

(2) For realisable global models, we construct realisations in terms of systems of local quotients. This technique differs from projections used, *e.g.*, for multi-party session types, where projections are partial operations depending on syntactic conditions (*cf.*, *e.g.*, [BY09]). Our approach assumes no restrictions on the form of global models. However, the syntactic restrictions used for global types guarantee some kind of communication properties of a resulting system which we consider separately (*cf.* Section 4).

(3) There are other papers in the literature in the context of different formalisms dealing with decomposition of port automata [KC09], Petri nets, or algebraic processes into (indecomposable) components [MM93, Lut16] used for the efficient verification and parallelisation of concurrent systems [CGM98, GM92] or to obtain better (optimised) implementations [TCV21, TCV22]. Moreover, Saeedloei and Kluźniak recently tackled the problem of realisability of sets of (sequential) timed scenarios as a timed scenario expression that realises the set [SK26].[9]

5.5. **Roadmap.**

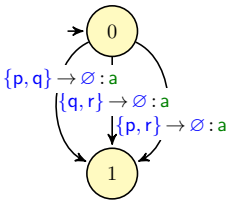
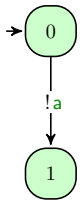
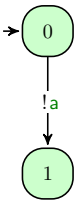
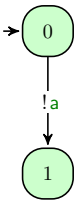
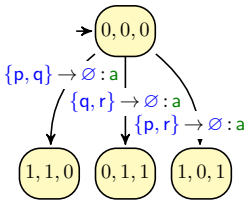
(1) Our current realisability approach does not deal with internal actions, which are however also an ingredient of the team automata framework and represented by system labels of the form (n, a) (*cf.* Section 2). They naturally appear when we build a team of CA which have internal actions. We believe, 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 5.6. 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 $\Lambda(\Theta, \mathbf{st}) = \{\{\mathsf{p}, \mathsf{q}\} \to \varnothing : \mathsf{a}, \{\mathsf{q}, \mathsf{r}\} \to \varnothing : \mathsf{a}, \{\mathsf{p}, \mathsf{r}\} \to \varnothing : \mathsf{a}\}$. 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).

[9] A timed scenario is a sequence of events with a set of constraints on the times at which the events occur.

TABLE 2. Global $\mathcal{M}$ violates $\mathbf{RC}(\mathcal{M},\equiv)$, but $\mathcal{S}=\{\mathcal{M}_{\mathsf{p}},\mathcal{M}_{\mathsf{q}},\mathcal{M}_{\mathsf{r}}\}$ realises $\mathcal{M}$

Global $\mathcal{M}$	Local $\mathcal{M}_{\mathsf{p}}$	Local $\mathcal{M}_{\mathsf{q}}$	Local $\mathcal{M}_{\mathsf{r}}$	$\mathbf{ta}(\mathcal{S},\mathbf{st})$
				

To see this, we compute the TA $\mathbf{ta}(\mathcal{S},\mathbf{st})$ 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 for both p and q because state 1 is indistinguishable from state 0 for both components p and q. Then we consider the interaction $\{\mathsf{p},\mathsf{q}\}\to\varnothing:\mathsf{a}$ one time for component p and starting in state $q_1=0$ and one time for component q and starting in state $q_2=0$. 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.

6. Composition of Systems

In this section, we consider how to compose systems of component automata and also how to compose synchronisation type specifications. We provide results concerning the preservation of communication properties. For this purpose we instantiate ideas presented in [tBHK20a, tBHP23a] by looking more closely to synchronisation type specifications and their composition instead of considering arbitrary synchronisation policies for team automata. Moreover, besides open system actions, we also allow internal actions here.

6.1. System Composition.

We assume given a finite set $\mathcal{K}$ of indices and a family $(\mathcal{S}_k)_{k\in\mathcal{K}}$ of systems $\mathcal{S}_k=(\mathcal{N}_k,(\mathcal{A}_n)_{n\in\mathcal{N}_k})$ with component automata $\mathcal{A}_n=(Q_n,q_{0,n},\Sigma_n,E_n)$ and component actions $\Sigma_n=\Sigma_n^?\cup\Sigma_n^!\cup\Sigma_n^\tau$ for each $n\in\mathcal{N}_k$. In particular, for each $k\in\mathcal{K}$, $Q_k=\prod_{n\in\mathcal{N}_k}Q_n$ is the set of *system states* of $\mathcal{S}_k$, $\Sigma_k=\bigcup_{n\in\mathcal{N}_k}\Sigma_n$ its set of *system actions*, $\Sigma_k^\bullet=\bigcup_{n\in\mathcal{N}_k}\Sigma_n^?\cap\bigcup_{n\in\mathcal{N}_k}\Sigma_n^!$ its set of *communicating actions*, and $\Sigma_k^\circ=\Sigma_k\setminus(\Sigma_k^\bullet\cup\bigcup_{n\in\mathcal{N}_k}\Sigma_n^\tau)$ its set of *open actions*.

Table 3. Lifting from systems to composed systems

	Systems	**Composition of systems**
Automata names	$\mathcal{N}_k$	$\mathcal{N}_\mathcal{K} = \bigcup_{n\in\mathcal{K}} \mathcal{N}_k$
System states	$Q_k = \prod_{n\in\mathcal{N}_k} Q_n$	$Q_\mathcal{K} = \prod_{n\in\mathcal{N}_\mathcal{K}} Q_n$
System actions	$\Sigma_k = \bigcup_{n\in\mathcal{N}_k} \Sigma_n$	$\Sigma_\mathcal{K} = \bigcup_{n\in\mathcal{N}_\mathcal{K}} \Sigma_n$
Communicating actions	$\Sigma_k^\bullet = \bigcup_{n\in\mathcal{N}_k} \Sigma_n^? \cap \bigcup_{n\in\mathcal{N}_k} \Sigma_n^!$	$\Sigma_\mathcal{K}^\bullet = \bigcup_{n\in\mathcal{N}_\mathcal{K}} \Sigma_n^? \cap \bigcup_{n\in\mathcal{N}_\mathcal{K}} \Sigma_n^!$
Open actions	$\Sigma_k^\circ = \Sigma_k \setminus (\Sigma_k^\bullet \cup \bigcup_{n\in\mathcal{N}_k} \Sigma_n^\tau)$	$\Sigma_\mathcal{K}^\circ = \Sigma_\mathcal{K} \setminus (\Sigma_\mathcal{K}^\bullet \cup \bigcup_{n\in\mathcal{N}_\mathcal{K}} \Sigma_n^\tau)$

To compose the system family $(\mathcal{S}_k)_{k\in\mathcal{K}}$ we assume that component names are unique and that system actions which are communicating in one system cannot occur in another system. Formally, the family $(\mathcal{S}_k)_{k\in\mathcal{K}}$ is *composable* if for all distinct $k, \ell \in \mathcal{K}$, $\mathcal{N}_k \cap \mathcal{N}_\ell = \varnothing$ and $\Sigma_k^\bullet \cap \Sigma_\ell = \varnothing$. In this case, we write $\mathcal{N}_\mathcal{K}$ for the disjoint union $\bigcup_{k\in\mathcal{K}} \mathcal{N}_k$. The *composition* of a composable family $(\mathcal{S}_k)_{k\in\mathcal{K}}$ of systems is defined as the system

$$\bigotimes_{k\in\mathcal{K}} \mathcal{S}_k = (\mathcal{N}_\mathcal{K}, (\mathcal{A}_n)_{n\in\mathcal{N}_\mathcal{K}}).$$

The state space of $\bigotimes_{k\in\mathcal{K}} \mathcal{S}_k$ is $Q_\mathcal{K} = \prod_{n\in\mathcal{N}_\mathcal{K}} Q_n$, the set of system actions of $\bigotimes_{k\in\mathcal{K}} \mathcal{S}_k$ is $\Sigma_\mathcal{K} = \bigcup_{n\in\mathcal{N}_\mathcal{K}} \Sigma_n$, the set of communicating actions of $\bigotimes_{k\in\mathcal{K}} \mathcal{S}_k$ is

$$\Sigma_\mathcal{K}^\bullet = \bigcup_{n\in\mathcal{N}_\mathcal{K}} \Sigma_n^? \cap \bigcup_{n\in\mathcal{N}_\mathcal{K}} \Sigma_n^!$$

and the set of open actions of $\bigotimes_{k\in\mathcal{K}} \mathcal{S}_k$ is $\Sigma_\mathcal{K}^\circ = \Sigma_\mathcal{K} \setminus (\Sigma_\mathcal{K}^\bullet \cup \bigcup_{n\in\mathcal{N}_\mathcal{K}} \Sigma_n^\tau)$. Table 3 lists the notions and notations of single systems and their lifted versions used for composed systems.

Obviously $\Sigma_k^\bullet \subseteq \Sigma_\mathcal{K}^\bullet$ for all $k \in \mathcal{K}$, *i.e.*, the communicating actions of the composed system contain the communicating actions of each of its subsystems. It is, however, important to notice that the composed system may have communicating actions which do not belong to the communicating actions of some subsystem, *i.e.*, the inclusion $\Sigma_k^\bullet \subseteq \Sigma_\mathcal{K}^\bullet$ can be strict for at least one $k \in \mathcal{K}$. This happens if a system action occurs as an open input action in (at least) one component of (at least) one subsystem and as open output action in (at least) one component of (at least) one *other* subsystem. Due to the composability conditions these actions cannot be communicating actions of a subsystem. They are called *interface actions* and formally defined by

$$\Sigma_{inf} = \Sigma_\mathcal{K}^\bullet \setminus \bigcup_{k\in\mathcal{K}} \Sigma_k^\bullet.$$

Of course, there might also remain system actions in $\Sigma_\mathcal{K}$ which are not in $\Sigma_\mathcal{K}^\bullet$ and also not internal to a component. Those actions still remain open for further system compositions. They are given by the set of open actions $\Sigma_\mathcal{K}^\circ$ defined above.

6.2. **Composition of STS.** The considerations so far did not consider synchronisation type specifications which are our crucial instrument to express which synchronisations are permitted in a system. Assume that each subsystem $\mathcal{S}_k$ of a composable family $(\mathcal{S}_k)_{k\in\mathcal{K}}$ comes with a synchronisation type specification $\mathbf{st}_k : \Sigma_k^\bullet \to \mathsf{Intv} \times \mathsf{Intv}$ (*cf.* Section 2). Due to the composability conditions, each single $\mathbf{st}_k$ can be uniquely extended to define a synchronisation type for each communicating action in $\bigcup_{k\in\mathcal{K}} \Sigma_k^\bullet$. Still missing are synchronisation types for the interface actions which must be provided by the system architect who composes the

family of systems. Formally, this is done by providing a function $\mathbf{st}_{inf} : \Sigma_{inf} \to \mathsf{Intv} \times \mathsf{Intv}$. This gives rise to an STS $\bigotimes_{k\in\mathcal{K}}^{\mathbf{st}_{inf}} \mathbf{st}_k : \Sigma_{\mathcal{K}}^{\bullet} \to \mathsf{Intv} \times \mathsf{Intv}$ for the composed system defined by:

$$\bigotimes_{k\in\mathcal{K}}^{\mathbf{st}_{inf}} \mathbf{st}_k(a) = \begin{cases} \mathbf{st}_k(a) & \text{if } a \in \Sigma_k^{\bullet} \text{ for some } k \in \mathcal{K} \\ \mathbf{st}_{inf}(a) & \text{if } a \in \Sigma_{inf} \end{cases}$$

Thus we can construct the team automaton $\mathbf{ta}(\bigotimes_{k\in\mathcal{K}} \mathcal{S}_k, \bigotimes_{k\in\mathcal{K}}^{\mathbf{st}_{inf}} \mathbf{st}_k)$. By construction, any transition of $\mathbf{ta}(\bigotimes_{k\in\mathcal{K}} \mathcal{S}_k, \bigotimes_{k\in\mathcal{K}}^{\mathbf{st}_{inf}} \mathbf{st}_k)$ is an extension of some transition in a team automaton $\mathbf{ta}(\mathcal{S}_k, \mathbf{st}_k)$ generated over some single system $\mathcal{S}_k$ by the STS $\mathbf{st}_k$. In particular, all synchronisation transitions using interface actions $a \in \Sigma_{inf}$ may extend the number of participants in accordance with the synchronisation type $\mathbf{st}_{inf}(a)$.[10] All other transitions[11] are just lifted from subsystems to the larger state space of $\bigotimes_{k\in\mathcal{K}} \mathcal{S}_k$.

As a consequence, projections of globally reachable states of $\mathbf{ta}(\bigotimes_{k\in\mathcal{K}} \mathcal{S}_k, \bigotimes_{k\in\mathcal{K}}^{\mathbf{st}_{inf}} \mathbf{st}_k)$ are reachable states in any team automaton $\mathbf{ta}(\mathcal{S}_k, \mathbf{st}_k)$. Therefore, we get that receptiveness and responsiveness requirements satisfied by the team automata of subsystems are also satisfied by the global team automaton. Moreover, if we have checked that the communication requirements for all interface actions $a \in \Sigma_{inf}$ with respect to $\mathbf{st}_{inf}(a)$ are satisfied by the global team automaton, it is ensured that the global team automaton is receptive and responsive. More details, including also the propagation of weak receptiveness and weak responsiveness, can be found in [tBHK20a]. In particular, for the preservation of weak communication properties one must be careful to avoid the introduction of cyclic waits.

Example 6.1. We consider two systems, $\mathcal{S}_{\mathsf{RaceV}}$ and $\mathcal{S}_{\mathsf{Arb}}$, depicted in Fig. 6. The system $\mathcal{S}_{\mathsf{RaceV}}$ is a variant of the Race system in Example 2.1. There are three component names R1, R2, and CtrlV. R1 and R2 are the names of the two runner components known from Fig. 1. The behaviour of the controller CtrlV is shown in the upper part of system $\mathcal{S}_{\mathsf{RaceV}}$ in Fig. 6. It is a variant of the controller in Fig. 1. The idea is that before the controller can start a race it must ask for a grant. The system $\mathcal{S}_{\mathsf{RaceV}}$ has two communicating actions, start and finish, and three open actions ask, grant, and reject. The system $\mathcal{S}_{\mathsf{Arb}}$, instead, has only one component, named Arbiter, whose behaviour is shown by the CA on the right side of Fig. 6. It is the task of the arbiter to grant or reject a race after being asked. The system $\mathcal{S}_{\mathsf{Arb}}$ has only open actions: ask, grant, and reject. We now compose the two systems resulting in the system $\mathcal{S}_{\mathsf{RaceV}} \otimes \mathcal{S}_{\mathsf{Arb}}$. The three actions ask, grant, and reject are the interface actions of the composed system (which is closed). All five actions start, finish, ask, grant, and reject are communicating actions of $\mathcal{S}_{\mathsf{RaceV}} \otimes \mathcal{S}_{\mathsf{Arb}}$. Synchronisation types for the communicating actions start and finish of the subsystem $\mathcal{S}_{\mathsf{RaceV}}$ are those provided in Example 2.2.

For the composition with $\mathcal{S}_{\mathsf{Arb}}$, we still have to provide synchronisation types for the interface actions ask, grant, and reject. We use binary communication for all of them, expressed by the synchronisation type ([1,1],[1,1]). Thus we have obtained an STS, which we abbreviate by $\mathbf{st}_{\otimes}$, for the composed system. The resulting team automaton for $\mathcal{S}_{\mathsf{RaceV}} \otimes \mathcal{S}_{\mathsf{Arb}}$, denoted by $\mathbf{ta}(\mathcal{S}_{\mathsf{RaceV}} \otimes \mathcal{S}_{\mathsf{Arb}}, \mathbf{st}_{\otimes})$, is a variant of the team automaton in Fig. 3. The difference appears in the first phase when the team automaton of the composed system performs the

[10] Transitions with open actions in subsystems that became interface actions but do not fit $\mathbf{st}_{inf}$ are omitted.

[11] These other transitions are transitions labelled with internal actions, communicating actions in a subsystem, and open actions in a subsystem which remain open in the composed system.

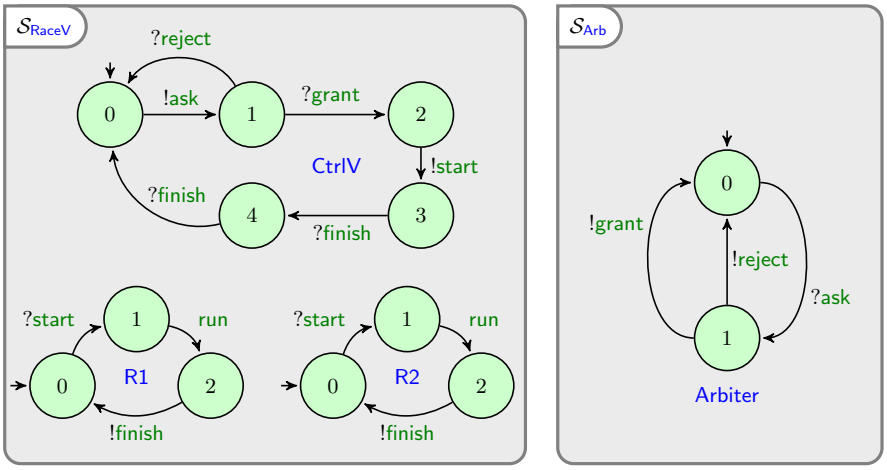


FIGURE 6. Two systems (adapted from [tBHK20c, tBHK20a])

interaction between the controller and the arbiter. This leads to two more states and three more transitions compared with Fig. 3.

To analyse communication properties we first point out that, similarly to Examples 4.1 and 4.2, the team automaton of the subsystem $\mathcal{S}_{\mathsf{RaceV}}$ is receptive and weakly responsive. By our preservation results from [tBHK20a] (Theorems 2 and 3), it remains to check that both properties are also satisfied for the three interface actions. Those actions can only occur in the initial phase of $\mathbf{ta}(\mathcal{S}_{\mathsf{RaceV}} \otimes \mathcal{S}_{\mathsf{Arb}}, \mathbf{st}_{\otimes})$. As an example, we consider the receptiveness requirement $\mathbf{rcp}(\mathsf{Ctrl}, \mathsf{ask})@(0,0,0,0)$, where all components are in their initial states and the controller wants to ask the arbiter for approval. Obviously, the TA $\mathbf{ta}(\mathcal{S}_{\mathsf{RaceV}} \otimes \mathcal{S}_{\mathsf{Arb}}, \mathbf{st}_{\otimes})$ is compliant with this requirement since, in its initial state, the arbiter is ready to receive the message. As an example of a responsiveness requirement, we consider $\mathbf{rsp}(\mathsf{Arbiter}, \mathsf{ask})@(0,0,0,0)$. It is clear that the TA $\mathbf{ta}(\mathcal{S}_{\mathsf{RaceV}} \otimes \mathcal{S}_{\mathsf{Arb}}, \mathbf{st}_{\otimes})$ is also compliant with this requirement since, in its initial state, the controller is ready to ask the arbiter. Now assume that the team automaton of the composed system is in state $(0,2,0,0)$, where the controller is in state 2, *i.e.*, it got the grant, and the arbiter is back to its initial state 0 (indicated by the first component of $(0,2,0,0)$). In such a state, the next possible action would be a start communication between the controller and the two runners. On the other hand, there is a responsiveness requirement of the arbiter expressed by $\mathbf{rsp}(\mathsf{Arbiter}, \mathsf{ask})@(0,2,0,0)$ saying that the arbiter expects an ask message. But this is not immediately possible, since the controller must first start and control a race. After that the controller will ask the arbiter again for getting a grant for the next race. Hence, the TA $\mathbf{ta}(\mathcal{S}_{\mathsf{RaceV}} \otimes \mathcal{S}_{\mathsf{Arb}}, \mathbf{st}_{\otimes})$ is weakly compliant with this responsiveness requirement. ▷

6.3. **Tool Support.** Our current tool (`http://arcatools.org/feta`) supports the composition of CA within a single system, with explicit external (input and output), and internal actions. However, there is no support yet to compose such systems.

6.4. **Related Work.** Composition of systems has also been studied for several related coordination models, such as Reo, BIP, and contract automata. Reo [Arb04] does not fully support systems, which are simply viewed as connectors. However, Reo supports the composition of connectors at its core, and further includes a hiding operator that restricts

which ports remain available for external interactions. Hence the composition of systems is at a different level with respect to the related formalisms.

In the framework of BIP, composition of components has been studied extensively in [GS05]. Composition is based on interaction models which must be respected by the interaction behaviour of sets of components. The spirit of our synchronisation types is similar, but BIP relies on connectors between ports with disjoint names instead of synchronisation types for shared actions. Differently from our communication properties, BIP studies preservation of different kinds of deadlock freedom.

In [BDF16], Basile *et al.* introduce two different operators for composing contract automata, representing two different orchestration policies. The first composition operator ($\otimes$, called *product* in [BDF16]) composes a singleton contract automaton $\mathcal{C}$ and a composite contract automaton $\mathcal{S}$ such that $\mathcal{C}$ cannot interact with the components of $\mathcal{S}$ but only with $\mathcal{S}$ as a whole. On the contrary, the second composition operator ($\boxtimes$, called *a-product* in [BDF16]) first extracts from its operands the singleton automata they are composed of, by projection, and then reassembles these according to the first composition operator. This means that matching request and offer actions in a composite contract automaton $\mathcal{S}$ become available for interaction with complementary actions in the components of the contract automaton it is composed with. Hence, for two singleton contract automata $\mathcal{C}_1$ and $\mathcal{C}_2$, $\mathcal{C}_1 \otimes \mathcal{C}_2 = \mathcal{C}_1 \boxtimes \mathcal{C}_2$. Moreover, while $\boxtimes$ is associative, $\otimes$ obviously is not. Composition of systems is not in the focus of *choreography automata* [BLT20, BLT23], whose theory is more related to questions of projection of global behaviour to local behaviour.

Multi-party session types, finally, offer parallel composition of systems expressed by $\mathcal{S} \parallel \mathcal{S}$. But, to the best of our knowledge, they are not concerned with preservation of communication safety by composition since, similarly to choreography automata, their focus is more on the relation between global types and their local projections.

6.5. **Roadmap.** Our composition operator is based on flattening system compositions such that the result is again a set of CA. From the software engineer's perspective we are also interested in hierarchical designs where sub-teams are first encapsulated into CA by hiding communicating actions. We believe this could simplify the analysis of behavioural properties of larger systems, *e.g.*, by using techniques of minimisation with respect to weak bisimulation. Furthermore, we intend to extend our tool to support the composition of systems.

Finally, it would be interesting to compare our approach to composition with the notion of so-called interleaving (weaving) composition, which uses a form of synchronisation based on assertions (*i.e.*, orthogonal to actions) and expresses directional (*i.e.*, rely-guarantee style) dependencies [BOV23]. According to the authors, these interleaving compositions are not characterisable by team automata synchronisations and—vice versa—cannot capture the full range of synchronisations offered by team automata.

7. Variability

We recently proposed featured team automata [tBCHP21a] to support *variability* in the development and analysis of teams. A featured team automaton can be projected into different variations, each from a given configuration determined by a selection of desired features that filter desired transitions; thus concisely capturing a family of concrete product (automata) models, as is common in software product line engineering [ABKS13].

Figure 7 shows the contributions of [tBCHP21a] in relation to earlier work [tBHK20a].

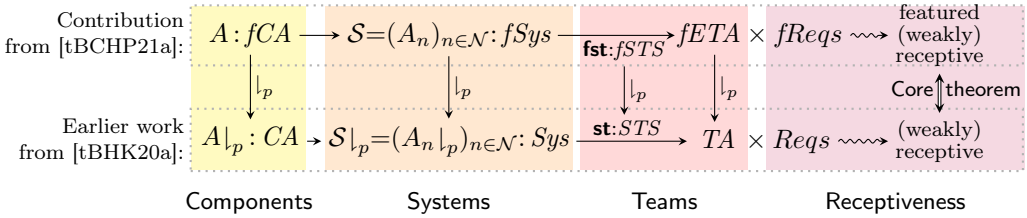
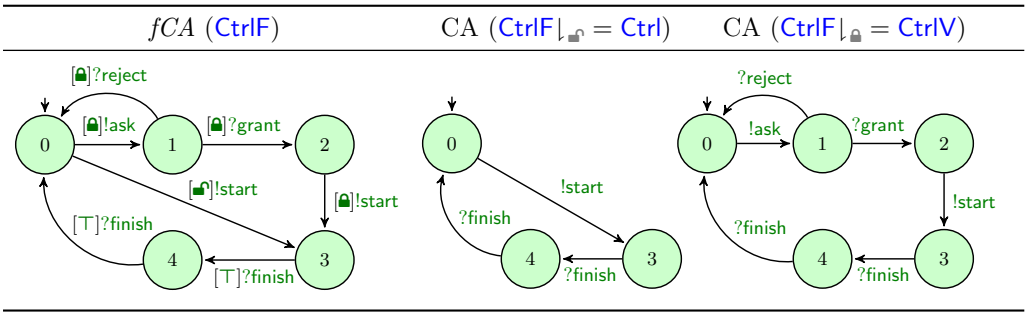


FIGURE 7. Overview of the contributions in our previous work [tBCHP21a] with respect to earlier work [tBHK20a], using a valid product p

In particular, we extended TA (as presented in this paper) by enriching them with variability, proposing a new model called *featured (extended) TA* (fETA) to allow the specification of—and reasoning on—a family of TA parameterised by a set of features. We define *projections* $\lfloor_p$ (for any valid product p) to relate the featured setting of [tBCHP21a] to that without in [tBHK20a]. We start by illustrating in the example below the notion of projection of featured component automata (fCA) to CA, covering the left part of Fig. 7 over Components.

Example 7.1. Recall the two versions of a controller: the original Ctrl (introduced in Fig. 1) and its variation CtrlV (introduced in Fig. 6) with a new approval phase before each race. Using a family model we can describe both variants in a single *superimposed* model—an *fCA*—presented on the left side of Table 4 (CtrlF). This *fCA* includes all possible transitions from both variants, each guarded by a Boolean formula over features (*e.g.*, [🔒]!ask is only active when the feature 🔒 is selected, [🔓]!start is only active when 🔓 is selected, and [⊤]?finish is always active). The semantics of CtrlF is given by the semantics of its *projections* to sets of features, each filtering the set of active transitions (*e.g.*, projecting CtrlF on the (singleton) set {🔓} yields the Ctrl component, while projecting it on the set {🔒} yields the CtrlV component, both depicted in Table 4). ▷

TABLE 4. Family model of a controller CtrlF and its two products obtained by projecting on the feature 🔓 (identical to Fig. 1) and on the feature 🔒 (identical to Fig. 6)



Example 7.1 illustrates how CA are projected in the Components part of the overview diagram in Fig. 7. This is trivially lifted for systems of CA (*cf.* the System part in Fig. 7), and required a notion of *featured* STS (synchronisation type specification) when enriching TA with variability (*cf.* the TA part in Fig. 7). Our core result related notions of receptiveness in fETA and its projections (*cf.* the Receptiveness part in Fig. 7), supporting a single analysis over families of systems (instead of over each possible projection).

The work on featured TA [tBCHP21a] describes a more complex example of a system, in which two users join and leave a server using two possible products: either with or without requiring permission from the server. The resulting superimposed featured system (*fSys*) counts 142 transitions and 18 states, which are reduced when building the fETA and projecting onto each TA, counting 10 transitions and 8 states with the simplest product.

This ability to present many variations of a state machine based on annotations of features over the transitions has been shown to be more efficient for model checking all variations in a single model (in a so-called *family-based* fashion [TAK+14]) rather than model checking each of the product variants individually (*cf.*, *e.g.*, [CHS+10, CCS+13, CDKB18, tBDL+22]).

7.1. **Featured Component Automata and Featured Systems.** We assume a finite set of *features* F, each regarded as a Boolean variable that represents a unit of variability. In our running example, $F = \{🔓, 🔒\}$.

A *product* is a finite combination of (desirable) features $p \subseteq F$. A *feature expression* ψ over a set of features F is a Boolean expression over features with the usual Boolean connectives and constants $\top$ and $\bot$ interpreted by the truth values true and false. A product p satisfies a feature expression ψ, denoted by $p \models \psi$, if and only if ψ is evaluated to $\top$ when $\top$ is assigned to every feature in p and $\bot$ to the features not in p. A feature expression ψ is *satisfiable* if there exists a product p such that $p \models \psi$.

A *feature model* fm is a feature expression that determines the set of products that satisfy the feature model.

A *featured transition system* (fTS) is a septuple $A = (Q, q_0, \Sigma, E, F, fm, \gamma)$[12] such that (Q, q_0, Σ, E) is an LTS, F is a finite set of features, fm is a feature model, and γ is a mapping assigning feature expressions to transitions in E. A product $p \subseteq F$ is *valid* for the feature model fm if $p \in fm$. A transition $t \in E$ is *realisable* for a valid product p if $p \models \gamma(t)$. An fTS A can be *projected* to a valid product $p \in fm$ by using γ to filter realisable transitions, resulting in the LTS $A\lfloor_p = (Q, I, \Sigma, E\lfloor_p)$, where $E\lfloor_p = \{\, t \in E \mid p \models \gamma(t) \,\}$.

A *featured component automaton* (fCA) is an fTS $A = (Q, q_0, \Sigma, E, F, fm, \gamma)$ such that $\Sigma = \Sigma^? \cup \Sigma^!$ are the disjoint sets of *input* and *output* actions as before. For simplicity, we do not consider internal actions here. Example 7.1 contains an fCA and its projections.

A *featured system* (fSys) is a pair $\mathcal{S} = (\mathcal{N}, (A_n)_{n\in\mathcal{N}})$, where $(A_n)_{n\in\mathcal{N}}$ is an $\mathcal{N}$-indexed family of fCA $A_n = (Q_n, q_{0,n}, \Sigma_n, E_n, F, fm, \gamma_n)$ over a shared set of features F and feature model fm. Any such fSys $\mathcal{S} = (\mathcal{N}, (A_n)_{n\in\mathcal{N}})$ induces an fTS $\mathsf{fts}(\mathcal{S}) = (Q, q_0, \Lambda, E, F, fm, \gamma)$, where $Q = \prod_{n\in\mathcal{N}} Q_n$ is the set of *system states*, writing q_n to represent the projection of a state $q \in Q$ to the fCA A_n, $q_0 = \prod_{n\in\mathcal{N}} q_{0,n}$ is the *initial system state*, Λ is the set of *system labels*, E is the set of *system transitions*, and γ holds the *system transition constraints*. Formally, Λ and E are defined as in Section 2, since they do not refer to features, and γ is built out of the individual γ_n by mapping each transition $t = q \xrightarrow{(\mathsf{out},a,\mathsf{in})} q' \in E$ to $\bigwedge_{n\in(\mathsf{out}\cup\mathsf{in})} \gamma_n(q_n \xrightarrow{a}_{A_n} q'_n)$. The *projection* of an fSys $\mathcal{S} = (\mathcal{N}, (A_n)_{n\in\mathcal{N}})$ on a product $p \in fm$ is the system $\mathcal{S}\lfloor_p = (\mathcal{N}, (A_n\lfloor_p)_{n\in\mathcal{N}})$.

[12] Throughout this section we use grey backgrounds to highlight extensions with features.

7.2. **Featured Team Automata.** Similarly to TA, fETA restrict the behaviour of (featured) systems with synchronisation types, here enriched with constraints over features and called *featured synchronisation type specification* (fSTS). An fSTS over an fSys $\mathcal{S}$, is a total function, $\mathsf{fst}\colon fm \times \Sigma \to \mathsf{Intv} \times \mathsf{Intv}$, mapping each product $p \in fm$ and action $a \in \Sigma$ to a traditional synchronisation type, *i.e.*, to a pair of intervals as before. Thus, an fSTS is parameterised by (valid) products and therefore supports variability of synchronisation conditions. For any product $p \in fm$, an fSTS fst can be projected on an STS $\mathsf{fst}\lfloor_p$ such that $\mathsf{fst}\lfloor_p(a) = \mathsf{fst}(p, a)$ for all $a \in \Sigma$.

Given an fSys $\mathcal{S} = (\mathcal{N}, (A_n)_{n\in\mathcal{N}})$ and an fSTS fst over $\mathcal{S}$, the *featured team automaton* (fETA) generated by $\mathcal{S}$ and fst is the fTS $\mathsf{fta}(\mathcal{S}, \mathsf{fst}) = (Q, q_0, \Lambda, E, F, fm, \gamma_{\mathsf{fst}})$, such that $\mathsf{fts}(\mathcal{S}) = (Q, q_0, \Lambda, E, F, fm, \gamma)$ is the fTS induced by $\mathcal{S}$, and γ_{fst} is a variation of γ that takes fst into account. Formally, for any $t \in E$, $\gamma_{\mathsf{fst}}(t)$ holds iff $\gamma(t)$ holds and if, for every valid product $p \in fm$, the senders and receivers of t are bounded by $\mathsf{fst}(p, t.a)$.

Receptiveness of fETA has been analysed in our previous work [tBCHP21a], corresponding to the right-most part of Fig. 7. Our core theorem shows that receptiveness of a family of TA can be checked by verifying, once and for all, receptiveness on the featured level. For doing so, we introduced a notion of featured receptiveness; *cf.* [tBCHP21a] for more details.

7.3. **Tool Support.** We implemented a prototypical tool to specify and analyse fETA. It can be used online and downloaded at `http://arcatools.org/feta`. It offers a DSL to describe a set of fCA, a feature model as a constraint, and an fSTS. The tool can calculate and draw the resulting fETA and the requirements for each state that must hold to guarantee receptiveness. It is implemented in Scala, using the Play Framework to generate an interactive website with a server that uses a Java SAT solver to reason over valid products.

7.4. **Related Work.** Variability has also been studied for related coordination models. BIP [KKW+16, MBBS16] has been extended with parameters to represent families of systems, one for each actual parameter, including approaches to model check such families. Contract automata [BDGG+17, BtBDGG17, BtBD+20] have also been lifted to a featured version, further investigating systems where the selected product can be updated at runtime. Reo [PC17] has been extended with numeric parameters describing the number of inputs and outputs of connectors, formalised using category theory. Petri nets [MPC11, MPC16] have been extended to featured nets, enriching arcs with conditions over features akin to fCA. I/O automata [LNW07, LPT09] have been lifted to product lines and analysed by model checking.

7.5. **Roadmap.** In the future, we intend to extend our theory of featured team automata to address receptiveness and responsiveness as well as compositionality, *i.e.*, extend fETA to composition of systems and investigate conditions under which communication safety is preserved by fETA composition. We would also like to lift to fETA our approaches to (1) model check communication requirements [tBCHP23], and (2) decompose a realisable global model into a system of CA [tBHP23a]. Moreover, we plan to further develop the tool and analyse the practical impact of fETA on the basis of larger case studies. This involves a thorough study of the efficiency of featured receptiveness checking compared to product-wise receptiveness checking. Finally, we would like to implement a family-based analysis algorithm that computes, for a given fETA, the set of all product configurations that yield communication-safe systems.

Table 5. Coordination formalisms and aspects analysed in this paper

Coordination Formalism (Sections 2 & 3)	Communication Properties (Section 4)	Realisation (Section 5)	Verification (Sections 4 & 5)	Composition of Systems (Section 6)	Variability (Section 7)	Supporting Tools (Sections 3 – 7)	Data
Team Automata [Ell97, tBEKR03]	✓	✓	✓	✓	✓	✓	
Reo via Port Automata [Arb04, KC09]			✓		✓	✓	✓
BIP (without priorities) [BBS06, BS08]			✓	✓	✓	✓	✓
Contract Automata [BDFT14, BDF16]	✓	✓	✓	✓	✓	✓	
Choreography Automata [BLT20,BLT23]	✓	✓	✓			✓	
Multi-Party Session Types [SY19, SD19]		✓	✓	✓	✓	✓	✓

8. Conclusion

We provided an overview of team automata, a model for capturing a variety of notions related to coordination in distributed systems with decades of history (*cf.* Sect. A) as witnessed by 25+ publications by 25+ researchers,[13] and discussed a number of aspects: communication properties, realisation, verification, composition of systems, variability, and tool support. Table 5 shows the relations between the main formalisms discussed in Section 3 (and included in the more detailed comparison in [tBHP24]) and the team automata aspects discussed in Sections 4–7, *i.e.*, *verification* of compliance of *communication properties* in terms of receptiveness and responsiveness, *verification* of the possible decomposition (*realisation*) of global models in terms of component models, *composition of systems* of component models, *variability* in the development and analysis of families of component models, and *tool support* for all these aspects). We did not discuss aspects like *data*, for which Reo and BIP provide native support, since team automata cannot currently deal with data.

In the past, we also studied in detail the computations and behaviour of team automata in relation to that of their constituting component automata [tBK03, tBK09], identifying several types of team automata that satisfy *compositionality* in terms of (synchronised) shuffles of their computations (*i.e.*, formal languages). Moreover, a process calculus for modelling team automata was proposed [tBGJ06, tBGJ08], extending some classical results on I/O automata as well as enlarging the family of team automata that guarantee a degree of compositionality. Team automata have also been used for the analysis of *security* aspects in communication protocols [tBLP03, tBLP05, tBLP06, EP06], in particular for spatial and spatio-temporal access control [tBEKR01b, NVK10].

In the future, we want to address the roadmaps identified in Sections 4.5, 5.5, 6.5, and 7.5. Recently, we have investigated *asynchronous* team automata [BtBP26], in which components use either FIFO channels or multisets to act as buffers for asynchronous communication, like asynchronous multi-party session types [HYC08] or systems of communicating finite state machines (CFSMs) [BZ83], rather than the synchronous communication based on simultaneous execution of shared actions used so far in TA and throughout the current paper. As a basis, we used CFSMs which, however, are based on peer-to-peer communication. This means that

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

it was challenging to understand how to generalise the approach by taking into account multi-action communication specified by synchronisation types. In such an asynchronous context, we obtained different kinds of communication properties and challenging investigations for their preservation by composition of systems (following, *e.g.*, ideas in the context of CFSMs from [BdH19,BH26]). We also provided tool support for modelling, animation, and automated checks for well-formedness and well-behavedness of asynchronous team automata [PtBB26].

Acknowledgements

We would like to thank the four reviewers for their careful reading and detailed comments, which have allowed us to considerably improve the 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 funded by National Funds through the FCT (Portuguese Foundation for Science and Technology) within the project POISE, with reference 2024.15783.PEX (https://doi.org/10.54499/2024.15783.PEX); by National Funds through FCT/MCTES, within the CISTER Unit (UIDP/UIDB/04234/2020); by the EU/Next Generation, within the Recovery and Resilience Plan, within project Route 25 (TRB/2022/00061 – C645463824-00000063); and by national funds through FCT and European funds through EU Chips JU, within project Shift2SDV (Grant Agreement No. 101194245).

Appendix A. Selected Publications from 25+ Years of Team Automata

Year	Title	Venue
2026 [BtBP26]	Asynchronous Team Automata	FM
2024 [tBHP24]	Team Automata: Overview and Roadmap	COORDINATION
2023 [tBHP23a]	Realisability of Global Models of Interaction	ICTAC
2023 [Pro23]	Overview on Constrained Multiparty Synchronisation in Team Automata	FACS
2023 [tBCHP23]	Can we Communicate? Using Dynamic Logic to Verify Team Automata	FM
2021 [tBCHP21a]	Featured Team Automata	FM
2020 [tBHK20c]	Team Automata@Work: On Safe Communication	COORDINATION
2020 [tBHK20a]	Compositionality of Safe Communication in Systems of Team Automata	ICTAC
2017 [tBCHK17]	Communication Requirements for Team Automata	COORDINATION
2016 [tBCK16]	Conditions for Compatibility of Components: The Case of Masters and Slaves	ISoLA
2014 [tBK14]	On Distributed Cooperation and Synchronised Collaboration	JALC
2013 [CK13]	Compatibility in a multi-component environment	TCS
2012 [tBK12]	Vector Team Automata	TCS
2010 [NVK10]	Team Automata Based Framework for Spatio-Temporal RBAC Model	BAIP
2009 [tBK09]	Associativity of Infinite Synchronized Shuffles and Team Automata	Fundam. Inform.
2008 [Sha08]	Extending Team Automata to Evaluate Software Architectural Design	COMPSAC
2008 [tBGJ08]	A calculus for team automata	ENTCS
2007 [SSAM07]	A Review on Specifying Software Architectures Using Extended Automata-Based Models	FSEN
2006 [EP06]	Modelling a Secure Agent with Team Automata	ENTCS
2006 [tBLP06]	A Team Automaton Scenario for the Analysis of Security Properties in Communication Protocols	JALC
2006 [tBGJ06]	A calculus for team automata	SBMF
2005 [tBLP05]	Team Automata for Security – A Survey –	ENTCS
2005 [tBK05]	Modularity for Teams of I/O Automata	IPL
2005 [Len05]	Integration of Analysis Techniques in Security and Fault-Tolerance (Chapter 6: *Security Analysis with Team Automata*)	PhD thesis
2005 [Pet05]	Aspects of Modeling and Verifying Secure Procedures (Chapter 4: *The Team Automata Chapter*)	PhD thesis
2004 [tBCVM04]	Teams of Pushdown Automata	IJCM
2004 [CK04]	Interactive Behaviour of Multi-Component Systems	Workshop ToBaCo
2003 [tBCVM03]	Teams of Pushdown Automata	PSI
2003 [tB03]	Team Automata: A Formal Approach to the Modeling of Collaboration Between System Components	PhD thesis
2003 [tBK03]	Team Automata Satisfying Compositionality	FME
2003 [tBLP03]	Team Automata for Security Analysis of Multicast/Broadcast Communication	Workshop WISP
2003 [Kle03]	Team Automata for CSCW – A Survey –	LNCS
2003 [tBEKR03]	Synchronizations in Team Automata for Groupware Systems	CSCW
2002 [EG02]	Towards Team-Automata-Driven Object-Oriented Collaborative Work	LNCS
2001 [tBEKR01b]	Team Automata for Spatial Access Control	ECSCW
2001 [tBEKR01a]	Team Automata for CSCW	Workshop
2000 [tHtB00]	A Conflict-Free Strategy for Team-Based Model Development	Workshop PDTSD
1999 [tBEKR99]	Synchronizations in Team Automata for Groupware Systems	Technical Report
1997 [Ell97]	Team Automata for Groupware Systems	GROUP

References

[ABB+18] Paul C. Attie, Saddek Bensalem, Marius Bozga, Mohamad Jaber, Joseph Sifakis, and Fadi A. Zaraket. Global and Local Deadlock Freedom in BIP. *ACM Trans. Softw. Eng. Methodol.*, 26(3):9:1–9:48, 2018. doi:10.1145/3152910.

[ABKS13] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer, 2013. doi:10.1007/978-3-642-37521-7.

[AG23] Muhammad Atif and Jan Friso Groote. *Understanding Behaviour of Distributed Systems Using mCRL2*. Springer, 2023. doi:10.1007/978-3-031-23008-0.

[AKM08a] Farhad Arbab, Natallia Kokash, and Sun Meng. Towards Using Reo for Compliance-Aware Business Process Modeling. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2008*, volume 17 of *CCIS*, pages 108–123. Springer, 2008. doi:10.1007/978-3-540-88479-8_9.

[AKM+08b] Farhad Arbab, Christian Krause, Ziyan Maraikar, Young-Joo Moon, and José Proença. Modeling, Testing and Executing Reo Connectors with the Eclipse Coordination Tools. Tool demo session of FACS 2008, 2008.

[Arb04] Farhad Arbab. Reo: a channel-based coordination model for component composition. *Math. Struct. Comput. Sci.*, 14(3):329–366, 2004. doi:10.1017/S0960129504004153.

[Arn94] A. Arnold. *Finite Transition Systems: Semantics of Communicating Systems*. Prentice Hall, 1994.

[BB20] Eduard Baranov and Simon Bliudze. Expressiveness of component-based frameworks: A study of the expressiveness of BIP. *Acta Inform.*, 57:761–800, 2020. doi:10.1007/s00236-019-00337-7.

[BBB+11] Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous Component-Based System Design Using the BIP Framework. *IEEE Softw.*, 28(3):41–48, 2011. doi:10.1109/MS.2011.27.

[BBS06] Ananda Basu, Marius Bozga, and Joseph Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *Proceedings of the 4th IEEE International Conference on Software Engineering and Formal Methods (SEFM 2006)*, pages 3–12. IEEE, 2006. doi:10.1109/SEFM.2006.27.

[BCD+08] Lorenzo Bettini, Mario Coppo, Loris D'Antoni, Marco De Luca, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. Global Progress in Dynamically Interleaved Multiparty Sessions. In Franck van Breugel and Marsha Chechik, editors, *Proceedings of the 19th International Conference Concurrency Theory (CONCUR 2008)*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008. doi:10.1007/978-3-540-85361-9_33.

[BCJ+15] Simon Bliudze, Alessandro Cimatti, Mohamad Jaber, Sergio Mover, Marco Roveri, Wajeb Saab, and Qiang Wang. Formal Verification of Infinite-State BIP Models. In Bernd Finkbeiner, Geguang Pu, and Lijun Zhang, editors, *ATVA 2015*, volume 9364 of *LNCS*, pages 326–343. Springer, 2015. doi:10.1007/978-3-319-24953-7_25.

[BČVZ06] Luboš Brim, Ivana Černá, Pavlína Vařeková, and Barbora Zimmerová. Component-Interaction Automata as a Verification-Oriented Component-Based System Specification. *ACM Softw. Eng. Notes*, 31(2), 2006. doi:10.1145/1118537.1123063.

[BCZ15] Massimo Bartoletti, Tiziana Cimoli, and Roberto Zunino. Compliance in Behavioural Contracts: A Brief Survey. In Chiara Bodei, Gian-Luigi Ferrari, and Corrado Priami, editors, *Programming Languages with Applications to Biology and Security*, volume 9465 of *LNCS*, pages 103–121. Springer, 2015. doi:10.1007/978-3-319-25527-9_9.

[BD24] Eike Best and Raymond Devillers. *Petri Net Primer: A Compendium on the Core Model, Analysis, and Synthesis*. Springer, 2024. doi:10.1007/978-3-031-48278-6.

[BDF16] Davide Basile, Pierpaolo Degano, and Gian-Luigi Ferrari. Automata for Specifying and Orchestrating Service Contracts. *Log. Meth. Comp. Sci.*, 12(4:6):1–51, 2016. doi:10.2168/LMCS-12(4:6)2016.

[BDFT14] Davide Basile, Pierpaolo Degano, Gian-Luigi Ferrari, and Emilio Tuosto. From Orchestration to Choreography through Contract Automata. In Ivan Lanese, Alberto Lluch Lafuente, Ana Sokolova, and Hugo Torres Vieira, editors, *Proceedings of the 7th Interaction and Concurrency Experience (ICE 2014)*, volume 166 of *EPTCS*, pages 67–85, 2014. doi:10.4204/EPTCS.166.8.

[BDFT16] Davide Basile, Pierpaolo Degano, Gian-Luigi Ferrari, and Emilio Tuosto. Relating two automata-based models of orchestration and choreography. *J. Log. Algebr. Methods Program.*, 85(3):425–446, 2016. doi:10.1016/J.JLAMP.2015.09.011.

[BDGG+17] Davide Basile, Felicita Di Giandomenico, Stefania Gnesi, Pierpaolo Degano, and Gian-Luigi Ferrari. Specifying Variability in Service Contracts. In *Proceedings of the 11th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2017)*, pages 20–27. ACM, 2017. doi:10.1145/3023956.3023965.

[BdH19] Franco Barbanera, Ugo de'Liguoro, and Rolf Hennicker. Connecting open systems of communicating finite state machines. *J. Log. Algebr. Methods Program.*, 109, 2019. doi:10.1016/j.jlamp.2019.07.004.

[BDL04] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A Tutorial on Uppaal. In Marco Bernardo and Flavio Corradini, editors, *Revised Lectures of the 4th International School*

on Formal Methods for the Design of Computer, Communication and Software Systems: Formal Methods for the Design of Real-Time Systems (SFM-RT 2004), volume 3185 of *LNCS*, pages 200–236. Springer, 2004. doi:10.1007/978-3-540-30080-9_7.

[BH26] Franco Barbanera and Rolf Hennicker. Safe orchestrated multicomposition of systems of communicating finite state machines. *J. Log. Algebr. Methods Program.*, 150:101109, 2026. doi:10.1016/J.JLAMP.2026.101109.

[BLT20] Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Choreography Automata. In Simon Bliudze and Laura Bocchi, editors, *Proceedings of the 22nd IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2020)*, volume 12134 of *LNCS*, pages 86–106. Springer, 2020. doi:10.1007/978-3-030-50029-0_6.

[BLT22] Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Formal Choreographic Languages. In Maurice H. ter Beek and Marjan Sirjani, editors, *Proceedings of the 24th IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2022)*, volume 13271 of *LNCS*, pages 121–139. Springer, 2022. doi:10.1007/978-3-031-08143-9_8.

[BLT23] Franco Barbanera, Ivan Lanese, and Emilio Tuosto. A Theory of Formal Choreographic Languages. *Log. Meth. Comp. Sci.*, 19(3):9:1–9:36, 2023. doi:10.46298/LMCS-19(3:9)2023.

[BMSH10] Sebastian S. Bauer, Philip Mayer, Andreas Schroeder, and Rolf Hennicker. On Weak Modal Compatibility, Refinement, and the MIO Workbench. In Javier Esparza and Rupak Majumdar, editors, *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2010)*, volume 6015 of *LNCS*, pages 175–189. Springer, 2010. doi:10.1007/978-3-642-12002-2_15.

[BMSZ17] Simon Bliudze, Anastasia Mavridou, Radoslaw Szymanek, and Alina Zolotukhina. Exogenous coordination of concurrent software components with JavaBIP. *Softw. Pract. Exper.*, 47(11):1801–1836, 2017. doi:10.1002/spe.2495.

[BOV23] Laura Bocchi, Dominic Orchard, and A. Laura Voinea. A Theory of Composing Protocols. *Art Sci. Eng. Program.*, 7(2):6:1–6:76, 2023. doi:10.22152/programming-journal.org/2023/7/6.

[BS08] Simon Bliudze and Joseph Sifakis. The Algebra of Connectors: Structuring Interaction in BIP. *IEEE Trans. Comput.*, 57(10):1315–1330, 2008. doi:10.1109/TC.2008.26.

[BSAR06] Christel Baier, Marjan Sirjani, Farhad Arbab, and Jan J. M. M. Rutten. Modeling component connectors in Reo by constraint automata. *Sci. Comput. Program.*, 61(2):75–113, 2006. doi:10.1016/J.SCICO.2005.10.008.

[BSBM05] Lucas Bordeaux, Gwen Salaün, Daniela Berardi, and Massimo Mecella. When are Two Web Services Compatible? In Ming-Chien Shan, Umeshwar Dayal, and Meichun Hsu, editors, *Proceedings of the 5th International Workshop on Technologies for E-Services (TES 2004)*, volume 3324 of *LNCS*, pages 15–28. Springer, 2005. doi:10.1007/978-3-540-31811-8_2.

[BtB22] Davide Basile and Maurice H. ter Beek. Contract Automata Library. *Sci. Comput. Program.*, 221, 2022. doi:10.1016/j.scico.2022.102841.

[BtB24] Davide Basile and Maurice H. ter Beek. Advancing orchestration synthesis for contract automata. *J. Log. Algebr. Methods Program.*, 2024.

[BtBD+20] Davide Basile, Maurice H. ter Beek, Pierpaolo Degano, Axel Legay, Gian-Luigi Ferrari, Stefania Gnesi, and Felicita Di Giandomenico. Controller synthesis of service contracts with variability. *Sci. Comput. Program.*, 187, 2020. doi:10.1016/j.scico.2019.102344.

[BtBDGG17] Davide Basile, Maurice H. ter Beek, Felicita Di Giandomenico, and Stefania Gnesi. 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)*, volume 2, pages 117–122. ACM, 2017. doi:10.1145/3109729.3109741.

[BtBL20] Davide Basile, Maurice H. ter Beek, and Axel Legay. Timed service contract automata. *Innov. Syst. Softw. Eng.*, 2(16):199–214, 2020. doi:10.1007/s11334-019-00353-3.

[BtBP20] Davide Basile, Maurice H. ter Beek, and Rosario Pugliese. 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. doi:10.23638/LMCS-16(2:9)2020.

[BtBP26] Davide Basile, Maurice H. ter Beek, and José Proença. Asynchronous Team Automata. In Augusto Sampaio and Mariëlle Stoelinga, editors, *Proceedings of the 27th International Symposium on Formal Methods (FM 2026)*, volume 16557 of *LNCS*, pages 27–49. Springer, 2026. doi:10.1007/978-3-032-26220-2_2.

[BvdBH+23] Simon Bliudze, Petra van den Bos, Marieke Huisman, Robert Rubbens, and Larisa Safina. JavaBIP meets VerCors: Towards the Safety of Concurrent Software Systems in Java. In Leen Lambers and Sebastián Uchitel, editors, *Proceedings of the 26th International Conference on Fundamental Approaches to Software Engineering (FASE 2023)*, volume 13991 of *LNCS*, pages 143–150. Springer, 2023. doi:10.1007/978-3-031-30826-0_8.

[BY09] Andi Bejleri and Nobuko Yoshida. Synchronous Multiparty Session Types. *Electr. Notes Theor. Comput. Sci.*, 241:3–33, 2009. doi:10.1016/j.entcs.2009.06.002.

[BZ83] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

[CC02] Josep Carmona and Jordi Cortadella. Input/Output Compatibility of Reactive Systems. In Mark Aagaard and John W. O'Leary, editors, *Proceedings of the 4th International Conference on Formal Methods in Computer-Aided Design (FMCAD 2002)*, volume 2517 of *LNCS*, pages 360–377. Springer, 2002. doi:10.1007/3-540-36126-X_22.

[CCS+13] Andreas Classen, Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, Axel Legay, and Jean-François Raskin. Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking. *IEEE Trans. Softw. Eng.*, 39(8):1069–1089, 2013. doi:10.1109/TSE.2012.86.

[CDKB18] Philipp Chrszon, Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. ProFeat: Feature-Oriented Engineering for Family-Based Probabilistic Model Checking. *Form. Asp. Comput.*, 30(1):45–75, 2018. doi:10.1007/s00165-017-0432-4.

[CDP12] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On Global Types and Multi-Party Sessions. *Log. Meth. Comp. Sci.*, 8(1):24:1–24:45, 2012. doi:10.2168/LMCS-8(1:24)2012.

[CDYP16] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *Math. Struct. Comput. Sci.*, 26(2):238–302, 2016. doi:10.1017/S0960129514000188.

[CGM98] Flavio Corradini, Roberto Gorrieri, and Davide Marchignoli. Towards parallelization of concurrent systems. *RAIRO Theor. Informatics Appl.*, 32(4-6):99–125, 1998. doi:10.1051/ita/1998324-600991.

[CGP09] Giuseppe Castagna, Nils Gesbert, and Luca Padovani. A Theory of Contracts for Web Services. *ACM Trans. Program. Lang. Syst.*, 31(5):19:1–19:61, 2009. doi:10.1145/1538917.1538920.

[CHS+10] Andreas Classen, Patrick Heymans, Pierre-Yves Schobbens, Axel Legay, and Jean-François Raskin. Model Checking Lots of Systems: Efficient Verification of Temporal Properties in Software Product Lines. In *Proceedings of the 32nd International Conference on Software Engineering (ICSE 2010)*, pages 335–344. ACM, 2010. doi:10.1145/1806799.1806850.

[CK04] Josep Carmona and Jetty Kleijn. Interactive Behaviour of Multi-Component Systems. In Jordi Cortadella and Alex Yakovlev, editors, *Proceedings of the ICATPN Workshop on Token-Based Computing (ToBaCo 2004)*, pages 27–31. University of Bologna, 2004.

[CK13] Josep Carmona and Jetty Kleijn. Compatibility in a multi-component environment. *Theor. Comput. Sci.*, 484:1–15, 2013. doi:10.1016/j.tcs.2013.03.006.

[Cla07] Dave Clarke. Coordination: Reo, Nets, and Logic. In Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever, editors, *Revised Lectures of the 6th International Symposium on Formal Methods for Components and Objects (FMCO 2007)*, volume 5382 of *LNCS*, pages 226–256. Springer, 2007. doi:10.1007/978-3-540-92188-2_10.

[CMT99] Ilaria Castellani, Madhavan Mukund, and P. S. Thiagarajan. Synthesizing Distributed Transition Systems from Global Specification. In C. Pandu Rangan, Venkatesh Raman, and Ramaswamy Ramanujam, editors, *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1999)*, volume 1738 of *LNCS*, pages 219–231. Springer, 1999. doi:10.1007/3-540-46691-6_17.

[CP18] Rúben Cruz and José Proença. ReoLive: Analysing Connectors in Your Browser. In Manuel Mazzara, Iulian Ober, and Gwen Salaün, editors, *Revised Selected Papers of the STAF 2018 Collocated Workshops*, volume 11176 of *LNCS*, pages 336–350. Springer, 2018. doi:10.1007/978-3-030-04771-9_25.

[DA18] Kasper Dokter and Farhad Arbab. Treo: Textual Syntax for Reo Connectors. In Simon Bliudze and Saddek Bensalem, editors, *Proceedings of the 1st International Workshop on Methods and Tools for Rigorous System Design (MeTRiD 2018)*, volume 272 of *EPTCS*, pages 121–135, 2018. doi:10.4204/EPTCS.272.10.

[dAH01] Luca de Alfaro and Thomas A. Henzinger. 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)*, pages 109–120. ACM, 2001. doi:10.1145/503209.503226.

[dAH05] Luca de Alfaro and Thomas A. Henzinger. Interface-Based Design. In *Engineering Theories of Software Intensive Systems*, volume 195 of *NATO Science Series*, pages 83–104. Springer, 2005. doi:10.1007/1-4020-3532-2_3.

[DGJ+15] Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jakšić, Jovanka Pantović, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. In Simon Gay and Jade Alglave, editors, *Proceedings of the 8th International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software (PLACES 2015)*, volume 203 of *EPTCS*, pages 29–43, 2015. doi:10.4204/EPTCS.203.3.

[DOS12] Francisco Durán, Meriem Ouederni, and Gwen Salaün. A generic framework for n-protocol compatibility checking. *Sci. Comput. Program.*, 77(7-8):870–886, 2012. doi:10.1016/j.scico.2011.03.009.

[EG02] Gregor Engels and Luuk P. J. Groenewegen. Towards Team-Automata-Driven Object-Oriented Collaborative Work. In Wilfried Brauer, Hartmut Ehrig, Juhani Karhumäki, and Arto Salomaa, editors, *Formal and Natural Computing*, volume 2300 of *LNCS*, pages 257–276. Springer, 2002. doi:10.1007/3-540-45711-9_15.

[EJPC24] Luc Edixhoven, Sung-Shik Jongmans, José Proença, and Ilaria Castellani. Branching pomsets: design, expressiveness and applications to choreographies. *J. Log. Algebr. Methods Program.*, 136, 2024. doi:10.1016/j.jlamp.2023.100919.

[Ell97] Clarence (Skip) Ellis. Team Automata for Groupware Systems. In *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge (GROUP 1997)*, pages 415–424. ACM, 1997. doi:10.1145/266838.267363.

[EP06] Lavinia Egidi and Marinella Petrocchi. Modelling a Secure Agent with Team Automata. *Electron. Notes Theor. Comput. Sci.*, 142:111–127, 2006. doi:10.1016/j.entcs.2004.12.046.

[GM92] Jan Friso Groote and Faron Moller. Verification of Parallel Systems via Decomposition. In Rance Cleaveland, editor, *Proceedings of the 3rd International Conference on Concurrency Theory (CONCUR 1992)*, volume 630 of *LNCS*, pages 62–76. Springer, 1992. doi:10.1007/BFb0084783.

[GM14] Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.

[GS05] Gregor Gössler and Joseph Sifakis. Composition for component-based modeling. *Sci. Comput. Program.*, 55:161–183, 2005. doi:10.1016/j.scico.2004.05.014.

[GT19] Roberto Guanciale and Emilio Tuosto. Realisability of pomsets. *J. Log. Algebr. Methods Program.*, 108:69–89, 2019. doi:10.1016/J.JLAMP.2019.06.003.

[HB18] Rolf Hennicker and Michel Bidoit. Compatibility Properties of Synchronously and Asynchronously Communicating Components. *Log. Meth. Comp. Sci.*, 14(1):1–31, 2018. doi:10.23638/LMCS-14(1:1)2018.

[HKT00] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. Foundations of Computing. MIT Press, 2000. doi:10.7551/mitpress/2516.001.0001.

[HT03] David Harel and P. S. Thiagarajan. Message Sequence Charts. In Luciano Lavagno, Grant Martin, and Bran Selic, editors, *UML for Real: Design of Embedded Real-Time Systems*, pages 77–105. Kluwer, 2003. doi:10.1007/0-306-48738-1_4.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2008)*, pages 273–284. ACM, 2008. doi:10.1145/1328438.1328472.

[ITU11] ITU (International Telecommunication Union). Message Sequence Chart (MSC). Recommendation ITU-T Z.120, Feb 2011. URL: http://www.itu.int/rec/T-REC-Z.120.

[JA12] Sung-Shik T. Q. Jongmans and Farhad Arbab. Overview of Thirty Semantic Formalisms for Reo. *Sci. Ann. Comput. Sci.*, 22(1):201–251, 2012. doi:10.7561/SACS.2012.1.201.

[Jon87] Bengt Jonsson. *Compositional Verification of Distributed Systems*. PhD thesis, Uppsala University, 1987.

[KC09] Christian Koehler and Dave Clarke. Decomposing Port Automata. In Sung Y. Shin and Sascha Ossowski, editors, *Proceedings of the 24th ACM Symposium on Applied Computing (SAC 2009)*, pages 1369–1373. ACM, 2009. doi:10.1145/1529282.1529587.

[KKdV10] Natallia Kokash, Christian Krause, and Erik P. de Vink. Data-Aware Design and Verification of Service Compositions with Reo and mCRL2. In *Proceedings of the 25th ACM Symposium on Applied Computing (SAC 2010)*, pages 2406–2413. ACM, 2010. doi:10.1145/1774088.1774590.

[KKSB11] Joachim Klein, Sascha Klüppelholz, Andries Stam, and Christel Baier. Hierarchical Modeling and Formal Verification: An Industrial Case Study Using Reo and Vereofy. In Gwen Salaün and Bernhard Schätz, editors, *Proceedings of the 16th International Workshop on Formal Methods for Industrial Critical Systems (FMICS 2011)*, volume 6959 of *LNCS*, pages 228–243. Springer, 2011. doi:10.1007/978-3-642-24431-5_17.

[KKW+16] Igor V. Konnov, Tomer Kotek, Qiang Wang, Helmut Veith, Simon Bliudze, and Joseph Sifakis. Parameterized Systems in BIP: Design and Model Checking. In Josée Desharnais and Radha Jagadeesan, editors, *Proceedings of the 27th International Conference on Concurrency Theory (CONCUR 2016)*, volume 59 of *LIPIcs*, pages 30:1–30:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.CONCUR.2016.30.

[KL98] Joost-Pieter Katoen and Lennard Lambert. Pomsets for Message Sequence Charts. In Yair Lahav, Adam Wolisz, Joachim Fischer, and Eckhardt Holz, editors, *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC (SAM 1998)*, pages 197–207. Humboldt-Universität zu Berlin, 1998.

[Kle03] Jetty Kleijn. Team Automata for CSCW – A Survey –. In Hartmut Ehrig, Wolfgang Reisig, Grzegorz Rozenberg, and Herbert Weber, editors, *Petri Net Technology for Communication-Based Systems: Advances in Petri Nets*, volume 2472 of *LNCS*, pages 295–320. Springer, 2003. doi:10.1007/978-3-540-40022-6_15.

[KLSV10] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Distributed Computing Theory. Springer, 2nd edition, 2010. doi:10.1007/978-3-031-02003-2.

[Len05] Gabriele Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. PhD thesis, University of Twente, 2005.

[LLN18] Kim Guldstrand Larsen, Florian Lorber, and Brian Nielsen. 20 Years of *Real* Real Time Model Validation. In Klaus Havelund, Jan Peleska, Bill Roscoe, and Erik P. de Vink, editors, *Proceedings of the 22nd International Symposium on Formal Methods (FM 2018)*, volume 10951 of *LNCS*, pages 22–36. Springer, 2018. doi:10.1007/978-3-319-95582-7_2.

[LNW07] Kim Guldstrand Larsen, Ulrik Nyman, and Andrzej Wąsowski. Modal I/O Automata for Interface and Product Line Theories. In Rocco De Nicola, editor, *Proceedings of the 16th European Symposium on Programming (ESOP 2007)*, volume 4421 of *LNCS*, pages 64–79. Springer, 2007. doi:10.1007/978-3-540-71316-6_6.

[LPT09] Kim Lauenroth, Klaus Pohl, and Simon Töhning. Model Checking of Domain Artifacts in Product Line Engineering. In *Proceedings of the 24th International Conference on Automated Software Engineering (ASE 2009)*, pages 269–280. IEEE, 2009. doi:10.1109/ASE.2009.16.

[LT89] Nancy A. Lynch and Mark R. Tuttle. An Introduction to Input/Output Automata. *CWI Q.*, 2(3):219–246, 1989. https://ir.cwi.nl/pub/18164.

[LT26] Nancy A. Lynch and Mark R. Tuttle. An Introduction to Input/Output Automata. *Form. Asp. Comput.*, 38(3):25:1–25:22, 2026. doi:10.1145/3788687.

[LTY15] Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From Communicating Machines to Graphical Choreographies. In *Proceedings of the 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2015)*, pages 221–232. ACM, 2015. doi:10.1145/2676726.2676964.

[Lut16] Bas Luttik. Unique parallel decomposition in branching and weak bisimulation semantics. *Theor. Comput. Sci.*, 612:29–44, 2016. doi:10.1016/j.tcs.2015.10.013.

[MBBS16] Anastasia Mavridou, Eduard Baranov, Simon Bliudze, and Joseph Sifakis. Architecture Diagrams: A Graphical Language for Architecture Style Specification. In Massimo Bartoletti, Ludovic Henrio, Sophia Knight, and Hugo Torres Vieira, editors, *Proceedings of the 9th Interaction and Concurrency Experience (ICE 2016)*, volume 223 of *EPTCS*, pages 83–97, 2016. doi:10.4204/EPTCS.223.6.

[MM93] Robin Milner and Faron Moller. Unique Decomposition of Processes. *Theor. Comput. Sci.*, 107(2):357–363, 1993. doi:10.1016/0304-3975(93)90176-T.

[MPC11] Radu Muschevici, José Proença, and Dave Clarke. Modular Modelling of Software Product Lines with Feature Nets. In Gilles Barthe, Alberto Pardo, and Gerardo Schneider, editors, *Proceedings of the 9th International Conference on Software Engineering and Formal Methods (SEFM 2011)*, volume 7041 of *LNCS*, pages 318–333. Springer, 2011. doi:10.1007/978-3-642-24690-6_22.

[MPC16] Radu Muschevici, José Proença, and Dave Clarke. Feature Nets: behavioural modelling of software product lines. *Softw. Sys. Model.*, 15(4):1181–1206, 2016. doi:10.1007/s10270-015-0475-z.

[NPW81] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri Nets, Event Structures and Domains, Part I. *Theor. Comput. Sci.*, 13:85–108, 1981. doi:10.1016/0304-3975(81)90112-2.

[NVK10] Jaisankar Narayanasamy, Sankaradass Veeramalai, and Arputharaj Kannan. Team Automata Based Framework for Spatio-Temporal RBAC Model. In Vinu V. Das, R. Vijayakumar, Narayan C. Debnath, Janahanlal Stephen, Natarajan Meghanathan, Suresh Sankaranarayanan, P. M. Thankachan, Ford Lumban Gaol, and Nessy Thankachan, editors, *Proceedings of the International Conference on Recent Trends in Business Administration and Information Processing (BAIP 2010)*, volume 70 of *CCIS*, pages 586–591. Springer, 2010. doi:10.1007/978-3-642-12214-9_106.

[ODPB+21] Simone Orlando, Vairo Di Pasquale, Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Corinne, a Tool for Choreography Automata. In Gwen Salaün and Anton Wijs, editors, *Proceedings of the 17th International Conference on Formal Aspects of Component Software (FACS 2021)*, volume 13077 of *LNCS*, pages 82–92. Springer, 2021. doi:10.1007/978-3-030-90636-8_5.

[PC17] José Proença and Dave Clarke. Typed connector families and their semantics. *Sci. Comput. Program.*, 146:28–49, 2017. doi:10.1016/j.scico.2017.03.002.

[PC20] José Proença and Guillermina Cledou. ARx: Reactive Programming for Synchronous Connectors. In Simon Bliudze and Laura Bocchi, editors, *Proceedings of the 22nd IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2020)*, volume 12134 of *LNCS*, pages 39–56. Springer, 2020. doi:10.1007/978-3-030-50029-0_3.

[PCdVA12] José Proença, Dave Clarke, Erik de Vink, and Farhad Arbab. Dreams: a framework for distributed synchronous coordination. In *Proceedings of the 27th ACM Symposium on Applied Computing (SAC 2012)*, pages 1510–1515. ACM, 2012. doi:10.1145/2245276.2232017.

[Pet05] Marinella Petrocchi. *Aspects of Modeling and Verifying Secure Procedures*. PhD thesis, University of Pisa, 2005.

[PM19] José Proença and Alexandre Madeira. Taming Hierarchical Connectors. In Hossein Hojjat and Mieke Massink, editors, *Revised Selected Papers of the 8th International Conference on Fundamentals of Software Engineering (FSEN 2019)*, volume 11761 of *LNCS*, pages 186–193. Springer, 2019. doi:10.1007/978-3-030-31517-7_13.

[Pro11] José Proença. *Synchronous Coordination of Distributed Components*. PhD thesis, Leiden University, 2011. URL: https://hdl.handle.net/1887/17624.

[Pro23] José Proença. Overview on Constrained Multiparty Synchronisation in Team Automata. In Javier Cámara and Sung-Shik Jongmans, editors, *Revised Selected Papers of the 19th International Conference on Formal Aspects of Component Software (FACS 2023)*, volume 14485 of *LNCS*, pages 194–205. Springer, 2023. doi:10.1007/978-3-031-52183-6_10.

[PtBB26] José Proença, Maurice H. ter Beek, and Davide Basile. A-Team artefact. Zenodo, 2026. doi:10.5281/zenodo.18784823.

[Rei13] Wolfgang Reisig. *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013. doi:10.1007/978-3-642-33278-4.

[RW87] Peter J. Ramadge and Walter M. Wonham. Supervisory Control of a Class of Discrete Event Processes. *SIAM J. Control Optim.*, 25(1):206–230, 1987. doi:10.1137/0325013.

[SD19] Paula Severi and Mariangiola Dezani-Ciancaglini. Observational Equivalence for Multiparty Sessions. *Fundam. Inform.*, 170(1-3):267–305, 2019. doi:10.3233/FI-2019-1863.

[Sha08] Mehran Sharafi. Extending Team Automata to Evaluate Software Architectural Design. In *Proceedings of the 32nd IEEE International Computer Software and Applications Conference (COMPSAC 2008)*, pages 393–400. IEEE, 2008. doi:10.1109/COMPSAC.2008.57.

[SK26] Neda Saeedloei and Feliks Kluźniak. Timed Scenario Expressions and Realisability. In Roberto Casadei and Fatemeh Ghassemi, editors, *Proceedings of the 28th IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2026)*, volume 16590 of *LNCS*, pages 3–25. Springer, 2026. doi:10.1007/978-3-032-28358-0_1.

[Sme18] Maarten Smeyers. A Browser-Based Graphical Editor for Reo Networks. Master's thesis, Leiden University, 2018. URL: https://theses.liacs.nl/1536.

[SSAM07] Mehran Sharafi, Fereidoon Shams Aliee, and Ali Movaghar. A Review on Specifying Software Architectures Using Extended Automata-Based Models. In Farhad Arbab and Marjan Sirjani, editors, *Proceedings of the 2nd International Symposium on Fundamentals of Software Engineering (FSEN 2007)*, volume 4767 of *LNCS*, pages 423–431. Springer, 2007. doi:10.1007/978-3-540-75698-9_30.

[SY19] Alceste Scalas and Nobuko Yoshida. Less Is More: Multiparty Session Types Revisited. *Proc. ACM Program. Lang.*, 3:30:1–30:29, 2019. doi:10.1145/3290343.

[TAK+14] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. A Classification and Survey of Analysis Strategies for Software Product Lines. *ACM Comput. Surv.*, 47(1):6:1–6:45, 2014. doi:10.1145/2580950.

[tB03] Maurice H. ter Beek. *Team Automata—A Formal Approach to the Modeling of Collaboration Between System Components*. PhD thesis, Leiden University, 2003. URL: https://hdl.handle.net/1887/29570.

[tBCHK17] Maurice H. ter Beek, Josep Carmona, Rolf Hennicker, and Jetty Kleijn. Communication Requirements for Team Automata. In Jean-Marie Jacquet and Mieke Massink, editors, *Proceedings of the 19th IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2017)*, volume 10319 of *LNCS*, pages 256–277. Springer, 2017. doi:10.1007/978-3-319-59746-1_14.

[tBCHP21a] Maurice H. ter Beek, Guillermina Cledou, Rolf Hennicker, and José Proença. Featured Team Automata. In Marieke Huisman, Corina Păsăreanu, and Naijun Zhan, editors, *Proceedings of the 24th International Symposium on Formal Methods (FM 2021)*, volume 13047 of *LNCS*, pages 483–502. Springer, 2021. doi:10.1007/978-3-030-90870-6_26.

[tBCHP21b] Maurice H. ter Beek, Guillermina Cledou, Rolf Hennicker, and José Proença. Featured Team Automata. Technical report, arXiv, Aug 2021. doi:10.48550/arXiv.2108.01784.

[tBCHP22a] Maurice H. ter Beek, Guillermina Cledou, Rolf Hennicker, and José Proença. Can we Communicate? Using Dynamic Logic to Verify Team Automata (Extended Version). Technical report, Zenodo, Dec 2022. doi:10.5281/zenodo.7418074.

[tBCHP22b] Maurice H. ter Beek, Guillermina Cledou, Rolf Hennicker, and José Proença. Can we Communicate? Using Dynamic Logic to Verify Team Automata (Software Artefact), 2022. doi:10.5281/zenodo.7338440.

[tBCHP23] Maurice H. ter Beek, Guillermina Cledou, Rolf Hennicker, and José Proença. Can we Communicate? Using Dynamic Logic to Verify Team Automata. In Marsha Chechik, Joost-Pieter Katoen, and Martin Leucker, editors, *Proceedings of the 25th International Symposium on Formal Methods (FM 2023)*, volume 14000 of *LNCS*, pages 122–141. Springer, 2023. doi:10.1007/978-3-031-27481-7_9.

[tBCK16] Maurice H. ter Beek, Josep Carmona, and Jetty Kleijn. Conditions for Compatibility of Components: The Case of Masters and Slaves. In Tiziana Margaria and Bernhard Steffen,

editors, *Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques (ISoLA 2016)*, volume 9952 of *LNCS*, pages 784–805. Springer, 2016. doi:10.1007/978-3-319-47166-2_55.

[tBCVM03] Maurice H. ter Beek, Erzsébet Csuhaj-Varjú, and Victor Mitrana. Teams of Pushdown Automata. In Manfred Broy and Alexandre V. Zamulin, editors, *Revised papers of the 5th International Andrei Ershov Memorial Conference on Perspectives of System Informatics (PSI 2003)*, volume 2890 of *LNCS*, pages 329–337. Springer, 2003. doi:10.1007/978-3-540-39866-0_32.

[tBCVM04] Maurice H. ter Beek, Erzsébet Csuhaj-Varjú, and Victor Mitrana. Teams of Pushdown Automata. *Int. J. Comput. Math.*, 81(2):141–156, 2004. doi:10.1080/00207160310001650099.

[tBDL+22] Maurice H. ter Beek, Ferruccio Damiani, Michael Lienhardt, Franco Mazzanti, and Luca Paolini. Efficient Static Analysis and Verification of Featured Transition Systems. *Empirical Software Engineering*, 22(1):10:1–10:43, 2022. doi:10.1007/s10664-020-09930-8.

[tBEKR99] Maurice H. ter Beek, Clarence A. Ellis, Jetty Kleijn, and Grzegorz Rozenberg. Synchronizations in Team Automata for Groupware Systems. Technical Report TR-99-12, Leiden Institute of Advanced Computer Science, Leiden University, 1999.

[tBEKR01a] Maurice H. ter Beek, Clarence A. Ellis, Jetty Kleijn, and Grzegorz Rozenberg. Team Automata for CSCW. In *Proceedings of the 2nd International Colloquium on Petri Net Technologies for Modelling Communication Based Systems*, pages 1–20. Fraunhofer ISST, 2001.

[tBEKR01b] Maurice H. ter Beek, Clarence A. Ellis, Jetty Kleijn, and Grzegorz Rozenberg. Team Automata for Spatial Access Control. In Wolfgang Prinz, Matthias Jarke, Yvonne Rogers, Kjeld Schmidt, and Volker Wulf, editors, *Proceedings of the 7th European Conference on Computer-Supported Cooperative Work (ECSCW 2001)*, pages 59–77. Kluwer, 2001. doi:10.1007/0-306-48019-0_4.

[tBEKR03] Maurice H. ter Beek, Clarence A. Ellis, Jetty Kleijn, and Grzegorz Rozenberg. Synchronizations in Team Automata for Groupware Systems. *Comput. Sup. Coop. Work*, 12(1):21–69, 2003. doi:10.1023/A:1022407907596.

[tBGJ06] Maurice H. ter Beek, Fabio Gadducci, and Dirk Janssens. A calculus for team automata. In Leila Ribeiro and Anamaria Martins Moreira, editors, *Proceedings of the 9th Brazilian Symposium on Formal Methods (SBMF 2006)*, pages 59–72. Istituto de Informatica da UFRGS, Porto Alegre, 2006.

[tBGJ08] Maurice H. ter Beek, Fabio Gadducci, and Dirk Janssens. A calculus for team automata. *Electron. Notes Theor. Comput. Sci.*, 195:41–55, 2008. doi:10.1016/j.entcs.2007.08.022.

[tBHK20a] Maurice H. ter Beek, Rolf Hennicker, and Jetty Kleijn. Compositionality of Safe Communication in Systems of Team Automata. In Violet Ka I Pun, Volker Stolz, and Adenilso Simão, editors, *Proceedings of the 17th International Colloquium on Theoretical Aspects of Computing (ICTAC 2020)*, volume 12545 of *LNCS*, pages 200–220. Springer, 2020. doi:10.1007/978-3-030-64276-1_11.

[tBHK20b] Maurice H. ter Beek, Rolf Hennicker, and Jetty Kleijn. Compositionality of Safe Communication in Systems of Team Automata. Technical report, Zenodo, Sep 2020. doi:10.5281/zenodo.4050293.

[tBHK20c] Maurice H. ter Beek, Rolf Hennicker, and Jetty Kleijn. Team Automata@Work: On Safe Communication. In Simon Bliudze and Laura Bocchi, editors, *Proceedings of the 22nd IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2020)*, volume 12134 of *LNCS*, pages 77–85. Springer, 2020. doi:10.1007/978-3-030-50029-0_5.

[tBHP23a] Maurice H. ter Beek, Rolf Hennicker, and José Proença. Realisability of Global Models of Interaction. In Erika Ábrahám, Clemens Dubslaff, and Silvia Lizeth Tapia Tarifa, editors, *Proceedings of the 20th International Colloquium on Theoretical Aspects of Computing (ICTAC 2023)*, volume 14446 of *LNCS*, pages 236–255. Springer, 2023. doi:10.1007/978-3-031-47963-2_15.

[tBHP23b] Maurice H. ter Beek, Rolf Hennicker, and José Proença. Realisability of Global Models of Interaction (Extended Version). Technical report, Zenodo, Sep 2023. doi:10.5281/zenodo.8377188.

[tBHP24] Maurice H. ter Beek, Rolf Hennicker, and José Proença. Team Automata: Overview and Roadmap. In Ilaria Castellani and Francesco Tiezzi, editors, *Proceedings of the 26th IFIP WG 6.1 International Conference on Coordination Models and Languages (COORDINATION 2024)*, volume 14676 of *LNCS*, pages 161–198. Springer, 2024. doi:10.1007/978-3-031-62697-5_10.

[tBK03] Maurice H. ter Beek and Jetty Kleijn. Team Automata Satisfying Compositionality. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *Proceedings of the 12th International Symposium of Formal Methods Europe (FME 2003)*, volume 2805 of *LNCS*, pages 381–400. Springer, 2003. doi:10.1007/978-3-540-45236-2_22.

[tBK05] Maurice H. ter Beek and Jetty Kleijn. Modularity for Teams of I/O Automata. *Inf. Process. Lett.*, 95(5):487–495, 2005. doi:10.1016/j.ipl.2005.05.012.

[tBK09] Maurice H. ter Beek and Jetty Kleijn. Associativity of Infinite Synchronized Shuffles and Team Automata. *Fundam. Inform.*, 91(3-4):437–461, 2009. doi:10.3233/FI-2009-0051.

[tBK12] Maurice H. ter Beek and Jetty Kleijn. Vector Team Automata. *Theor. Comput. Sci.*, 429:21–29, 2012. doi:10.1016/j.tcs.2011.12.020.

[tBK14] Maurice H. ter Beek and Jetty Kleijn. On Distributed Cooperation and Synchronised Collaboration. *J. Autom. Lang. Comb.*, 19(1-4):17–32, 2014. doi:10.25596/jalc-2014-017.

[tBLP03] Maurice ter Beek, Gabriele Lenzini, and Marinella Petrocchi. Team Automata for Security Analysis of Multicast/Broadcast Communication. In Nadia Busi, Roberto Gorrieri, and Fabio Martinelli, editors, *Proceedings of the ICATPN Workshop on Issues in Security and Petri Nets (WISP 2003)*, pages 57–71. Eindhoven University of Technology, 2003.

[tBLP05] Maurice H. ter Beek, Gabriele Lenzini, and Marinella Petrocchi. Team Automata for Security – A Survey –. *Electron. Notes Theor. Comput. Sci.*, 128:105–119, 2005. doi:10.1016/j.entcs.2004.11.044.

[tBLP06] Maurice H. ter Beek, Gabriele Lenzini, and Marinella Petrocchi. A Team Automaton Scenario for the Analysis of Security Properties in Communication Protocols. *J. Autom. Lang. Comb.*, 11(4):345–374, 2006. doi:10.25596/jalc-2006-345.

[TCV21] Viktor Teren, Jordi Cortadella, and Tiziano Villa. Decomposition of transition systems into sets of synchronizing state machines. In *Proceedings of the 24th Euromicro Conference on Digital System Design (DSD 2021)*, pages 77–81. IEEE, 2021. doi:10.1109/DSD53832.2021.00021.

[TCV22] Viktor Teren, Jordi Cortadella, and Tiziano Villa. 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)*, pages 165–173. IEEE, 2022. doi:10.1109/DSD57027.2022.00031.

[tHtB00] Pieter Jan 't Hoen and Maurice H. ter Beek. A Conflict-Free Strategy for Team-Based Model Development. In *Proceeedings of the International Workshop on Process support for Distributed Team-based Software Development (PDTSD 2000)*, pages 720–725. IIS, 2000.

[vBvES94] Johan van Benthem, Jan van Eijck, and Vera Stebletsova. Modal Logic, Transition Systems and Processes. *J. Log. Comput.*, 4(5):811–855, 1994. doi:10.1093/logcom/4.5.811.

[vO02] David von Oheimb. Interacting State Machines: A Stateful Approach to Proving Security. In Ali E. Abdallah, Peter Y. A. Ryan, and Steve A. Schneider, editors, *Revised Papers of the 1st International Conference on Formal Aspects of Security (FASec 2002)*, volume 2629 of *LNCS*, pages 15–32. Springer, 2002. doi:10.1007/978-3-540-40981-6_4.

[Win88] Glynn Winskel. An introduction to event structures. In Jaco W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency: Proceedings of the Research and Education in Concurrent Systems Workshop (REX 1988)*, volume 354 of *LNCS*, pages 364–397. Springer, 1988. doi:10.1007/BFB0013026.

[Yos24] Nobuko Yoshida. Programming Language Implementations with Multiparty Session Types. In Frank S. de Boer, Ferruccio Damiani, Reiner Hähnle, Einar Broch Johnsen, and Eduard Kamburjan, editors, *Active Object Languages: Current Research Trends*, volume 14360 of *LNCS*, pages 147–165. Springer, 2024. doi:10.1007/978-3-031-51060-1_6.

This work is licensed under the Creative Commons Attribution License. To view a copy of this license, visit https://creativecommons.org/licenses/by/4.0/ or send a letter to Creative Commons, 171 Second St, Suite 300, San Francisco, CA 94105, USA, or Eisenacher Strasse 2, 10777 Berlin, Germany