Overview on Constrained Multiparty Synchronisation in Team Automata

José Proença

CISTER and University of Porto, Portugal, `jose.proenca@fc.up.pt`

Abstract. This paper provides an overview on recent work on Team Automata, whereby a network of automata interacts by synchronising actions from multiple senders and receivers. We further revisit this notion of synchronisation in other well known concurrency models, such as Reo, BIP, Choreography Automata, and Multiparty Session Types.

We address realisability of Team Automata, i.e., how to infer a network of interacting automata from a global specification, taking into account that this realisation should satisfy exactly the same properties as the global specification. In this analysis we propose a set of interesting directions of challenges and future work in the context of Team Automata or similar concurrency models.

1 Introduction

Many different formal models for concurrent systems exist, each with its own advantages and disadvantages. This short paper provides an overview on recent work on *Team Automata* (TA), and takes a step back to relate the *constrained multiparty synchronisation* of TA with other popular models in this community studying fundamentals of component-based software.

TA were initially proposed by ter Beek et al. [9], inspired by similar approaches such as I/O automata [31] and interface automata [23], whereby a network of automata with input and output labels interacts. This interaction in TA involves *multiparty synchronisations*, whereas multiple senders and receivers can participate in a single atomic global transition. It is also *constrained* because it is parameterised by a synchronisation policy that, for each label, specifies the possible number of senders and of receivers that it can have.

Composing a network of components in a team, using constrained multiparty synchronisation, yields a new transition system whose labels are interactions consisting of (i) a message name or type, (ii) a set of components that send this message, and (iii) a set of components that receive this message. Only valid interactions are allowed, i.e., that obey the associated synchronisation policy. Many topics have been investigated in the context of TA since 2003, including security [15,17], composition and expressivity [10,13], variability [8], and compatibility of components [6,7,11,22]. Labels of early versions of TA include only message names, and were later extended to explicitly capture which components participate in system transitions [11], giving rise to a concurrent semantics. This
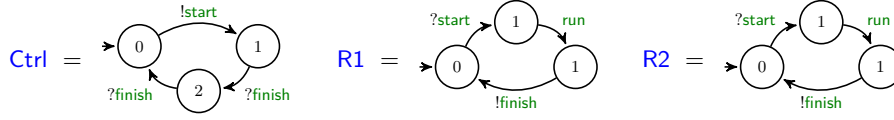
Fig. 1: Race example: a controller asks simultaneously 2 runners to start and receives a finish message once each is of them is done

ideas has been also used in Vector TA [14], where actions include active participants. In this paper we use these extended labels in TA.

We revisit several kinds of interactions in the *orchestration models* Reo [29] and BIP [20], and attempt to frame this multiparty synchronisation in the context of *choreographic models* such as choreographic automata [3] and (synchronous) multiparty session types [36]. We further address realisation of TA, i.e. how to obtain a set of interacting components from a global transition system over interactions, given the synchronisation policy. This last part is ongoing work [12], in collaboration with Rolf Hennicker and Maurice ter Beek, and include a detour on how to specify properties of interest, and how to guarantee that these are preserved when realising a global specification.

The explanations in this paper are relatively informal, driven mainly by examples. We proceed by introducing an example used throughout this paper.

Motivating example We use as a running example a Race system (Fig. 1), borrowed from previous work [16], consisting of 3 communicating components: a controller Ctrl and two runners R1, R2. Each is an automaton with input actions (?start and ?finish), output actions (!start and !finish), and internal actions (run). Interactions are subject to the following synchronisation policy: (i) the 3 actions start must synchronise, i.e., must be performed atomically, representing a simultaneous start of both runners; and (ii) the 3 actions finish must synchronise in pairs, i.e., each runner should atomically notify the controller that s/he finished, but the controller must receive each message one at a time. The internal action run is not involved in any interaction.

The composition of these 3 automata combined with the synchronisation restrictions of the start and finish actions yields what we call a team automaton [5, 26]. Labels of our team automaton are *interactions* involving the senders, the receivers, and the action name. E.g., "Ctrl → {R1, R2} : start" is an interaction that labels a transition in our team automaton, which follows our synchronisation policy stating that start should always have one sender and two receivers.

Many studies on TA investigate whether the components of a system will ever fail when trying to send a particular message (receptiveness), or to receive a set of possible messages (responsiveness) [6, 8, 10]. Another topic of interest addressed in this paper, in the context of TA or other similar systems, regards realisability or decomposition, i.e., whether these 3 components can be discovered given a description of the global behaviour capturing the same behaviour.

Organisation of the paper Sect. 2 investigates how the multiparty synchronisation of our Race example can be captured by other formalisms in the literature, considering both orchestration and choreographic languages. Sect. 3 addresses

2

how to specify properties of interest for TA and proposes ideas and challenges regarding the realisation of TA. Sect. 4 concludes this paper.

2 Related models

This section provides an overview of alternative approaches to model our race example with existing models that describe interaction patterns. We start with *orchestration* models, where a naive implementation of the interaction patterns leads to a centralised component controlling the interactions. We continue with *choreographic* models, which focus on how to describe the interaction *contracts* that each computational component must obey to derive an intended communication protocol, without relying on a dedicated component with the interaction logic. The precise distinction between orchestration and choreographic models is not consensual among researchers, whereas one can provide arguments why a given orchestration model can be considered to be a choreographic one and vice-versa.

Unlike the models listed below, TA identify which actions from composed components can interact by considering (1) the name of the action, (2) the direction of dataflow, and (3) the synchronisation policy (i.e., for each action, how many inputs and outputs are required or allowed). Some models below assume that all names are distinct, and that the interaction models must relate them (e.g., as an enumeration of possible sets of actions that must synchronise), sometimes leading to a more exhaustive description of the possible interactions. Other models below require interactions to be always 1-to-1 (called peer-to-peer in [9]), possibly requiring more intermediate components to capture the intended behaviour.

2.1 Orchestration

Models and languages that orchestrate components focus on how to group the interaction logic in a connector, restricting how the components can interact. This section analyses our race example in the context of the Reo [29] and the BIP [20] coordination models, since they focus on the analysis of interactions of systems with synchronous interactions that involve multiple participants. These have been compared in more detailed by Dokter et al. [24].

Reo Reo is a graphical modelling to orchestrate components based on the synchronous composition of a set of primitive connectors [29]. Many different semantic formalisms exist to provide a semantics to Reo connectors – we focus here on constraint automata [2].

We encode our race example as a Reo connector on the left side of Fig. 2, and its semantics given by a constraint automaton on the right side. This semantics (and most Reo semantics) is stateful, i.e., the next set of possible interactions depends on the previously taken interactions. Reo is highly compositional, in the sense that the global interaction patterns result from composing simpler connectors. In our example from Fig. 2, we count 2 FIFO channels (⊣☐➤), 2
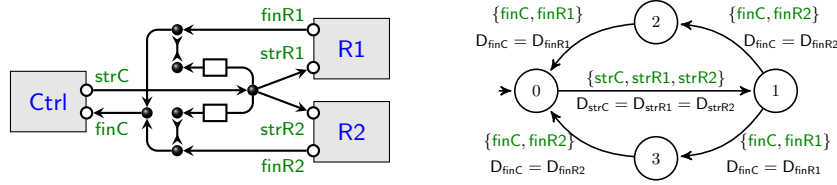
Fig. 2: Reo approach: the connector (left) built compositionally describing valid sequences of interactions from participants, and its semantics given by a constraint automata (right)

synchronous barriers (>——◁), 3 replicators (→•◁ after strC, finR1, and finR2), and 1 interleaving merger (⋝•→ before finC). Each of these has a defined semantics given by constraint automata [2], and composing these 8 automata (and hiding internal names) yields the constraint automata on the right of Fig. 2. Here each arc is labelled with a set of ports that must synchronise and a constraint over the data that must flow in each port. The direction of dataflow is not captured by constraint automata.

Reo focuses on the connector and not on the components. This means that the connector dictates which ports are active and inactive at each moment, allowing only desirable patterns of interaction. Hence, the precise behaviour of a runner, e.g., is not explicit in this model, including other internal actions that s/he may perform. When connecting a concrete runner to this connector one should verity whether it is compliant, i.e., if its possible patterns of interaction are consistent with the patterns imposed by the connector. This compliance check can be regarded as a type check against a behavioural type, as used in Session Types, described in the next section, and investigated less in the context of Reo.

BIP BIP [20] is a language to specify the expected behaviour of components and of component architectures. A program in BIP consists of a labelled state machine for each component (**B**ehaviour), a set of possible synchronous **I**nteractions between transitions of different components, and a possible partial order among interactions (**P**riority). Semantic models, such as the algebra of connectors [20], formalise the semantics of interactions.

Our race example is modelled in BIP in Fig. 3. Similarly to TA, the components describe the core behaviour of a system, here represented as labelled transition systems (also 1-safe Petri Nets can be used in some tools). The set of valid interactions is stateless, and are depicted by connecting ports that must synchronise; the dotted connection is an alternative to the its solid counterpart, connecting R2.finish instead of R1.finish. Trigger ports, not used here, can also be used instead of rendez-vous ports (●) to denote broadcasts, i.e., all siblings of a trigger in an interaction that are ready to synchronise will do so, and the trigger will always succeed.

TA share some concepts with BIP: the behaviour of the components and some synchronisation restrictions are modelled separately. Unlike TA, there is no explicit direction of dataflow. Interactions are often enriched with descriptions of how the dataflow should be updated, meaning that different connectors may

treat the same ports as inputs or outputs. Unlike BIP, TA do not support any combination of synchronising ports, but only those that share the same name.

2.2 Choreographies

We consider a choreographic model to be a language or a calculus that describes the global set of valid interactions. Each interaction is typically described by (i) the name that identifies the interface, (ii) the sender, and (iii) the receiver. This richer representation of interactions, with respect to the ones used in the previous subsection, allows the behaviour of each participant to be derived from the global model, which is often the ultimate goal of these models. Most of these support only a single sender and a single receiver, and are not meant for multiparty synchronisations, although it is possible to lift many of these models to the latter case. In contrast, Reo is typically agnostic to the concrete components connected to the ports of a connector, and describes only their valid patterns of interaction, and BIP avoids specifying explicitly the global behaviour (focusing on the local behaviour).

This section provides an overview of choreographic automata [33] to illustrate an approach to reason over deterministic automata in these rich interactions, and of synchronous multiparty session types [36]. When modelling our Race example we use a variation that allows multiple senders and receivers, without providing its formal semantics.

Choreographic Automata (CA) CA [3] are automata with labels that describe interactions, including a sender, a receiver, and a message name.

Our race example is encoded as a variation of CA on the left of Fig. 4, where we use a set of senders and receivers at each interaction to match the semantics of our composed TA. For comparison, the composed TA of our running example is depicted on the right of Fig. 4, where each state is a triple with the states of the local components, and the labels are interactions following the notation in [11]. We drop the curly brackets of singleton sets in the CA for simplicity. We also avoid representing the internal action run, although this could have been done
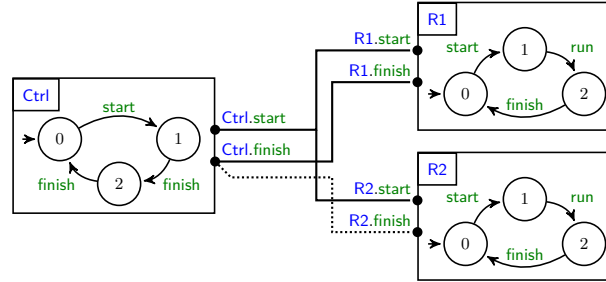

Fig. 3: BIP approach: individual components are labelled by actions, which are subject to the set of valid rendez-vous interaction constraints imposed by the middle (stateless) connector

5

with our extension to sets of senders and receivers, e.g., writing $\{\mathsf{R1}\} \to \{\}$: run to denote a send by $\mathsf{R1}$ with no receivers. This would have resulted in a larger set of states capturing possible interleavings.

Multiparty Session Types (MPST) MPST use a calculus to describe global behaviour. We can represent our race example using a variation of the global type of a synchronous MPST [28, 36] as follows.

$$\lambda X \cdot \mathsf{Ctrl} \to \{\mathsf{R1}, \mathsf{R2}\} : \left\{ \mathsf{start}.\begin{pmatrix} \mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}. \\ \mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish}.X \end{pmatrix}, \mathsf{start}.\begin{pmatrix} \mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish}. \\ \mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}.X \end{pmatrix} \right\}$$

This example uses a fixed point (λX) to iterate [28]. Interactions are written as $\{\mathsf{A}\} \to \{\mathsf{B}\} : \{\mathsf{msg}_i.C_i\}_{i \in \mathsf{I}}$, for any sets of participants A, B and I, messages msg_i, and continuations C_i of the choreography; we omit curly brackets when there is only one element. The set of messages and continuations denotes the choices after communications.

This example does not follow the usual MPST syntax: it uses a set of receivers in the first interaction, and includes two choices of messages inside the big curly brackets that start with the same message start. Most work on MPST supports only a single sender and receiver in each interaction, and choices with syntactic restrictions, such as the need to distinguish the first message. Up to our knowledge, without these extensions and without introducing new messages the Race example cannot be faithfully modelled.

A global choreography of our reference synchronous MPST [28, 36], to be valid, must obey several properties, including the need to have different messages at the start of each choice, and it must be possible to produce a projection for each participant whereas each participant does not (syntactically) distinguish choices made by other participants.

3 Behavioural properties and realisability

Having a TA (or another model of constrained interactions) allows us to reason over desired behavioural properties. Our properties of interest can target either a **specific** scenario, e.g., that a runner must always run before finishing, or be more **general**, e.g., no component can ever fail to send or receive messages. Hence we believe that *dynamic logic* provides the right level of abstraction to describe valid sequences of actions, which excels at capturing what actions can
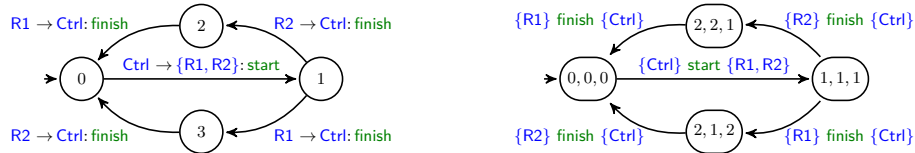

Fig. 4: Choreographic Automaton (left), extended with multiple senders and receivers, and composed TA (right) of the automata in Fig. 1

6

and cannot be performed throughout an execution. Other alternatives, such as traditional linear time logic (LTL) and computation tree logic (CTL), often focus on reachability of states and not on sequences of actions, making these less optimal for our properties of interest.

A simpler alternative to dynamic logics could be the use of regular expressions to model valid patterns of interactions. This is aligned with existing approaches such as CA that focus on the languages accepted by these automata [4] to reason over properties of such systems. The choice of using dynamic logics, regular expressions, or other model to specify properties, impacts realisability, since it is desirable for global models and their realisations to satisfy the same properties.

Many model checkers exist to verify temporal properties of concurrent systems, including mCRL2 [21] and Uppaal [18]. These support either dynamic logic (mCRL2) or CTL. Both Reo and TA have been encoded as mCRL2 processes to exploit its powerful model-checking engine [16,30,35]. The notion of synchronisation types, describing the number of possible senders and receivers by each message, can be captured using mCRL2, although it requires constructing the set of all concrete combinations of ports that can synchronise. This notion cannot directly be mapped to Uppaal, which supports only pairwise communication or a form of broadcast that differs from the one in TA (c.f. [9]).

3.1 Propositional Dynamic Logic

In our Race example it is desirable that, for any runner that starts running, it should be possible to finish her/his run. This can be expressed by the dynamic logic formula below, using regular expressions over interactions as actions, and writing *some* to denote the non-deterministic choice over any interaction:

$$\left[some^*; \mathsf{Ctrl} \to \{\mathsf{R1},\mathsf{R2}\} : \mathsf{start} \right] \begin{pmatrix} \langle some^*; \mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish} \rangle \; true \; \wedge \\ \langle some^*; \mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish} \rangle \; true \end{pmatrix}$$

Informally, $[\alpha]\psi$ holds if, after any of the sequence of interactions covered by the regular expression α, ψ holds. Similarly, $\langle \alpha \rangle \psi$ holds if it is possible to perform any of the sequence of interactions covered by the regular expression α, and end up in a state where ψ holds. The formula above means that, after any sequence of interactions that ends in $\mathsf{Ctrl} \to \{\mathsf{R1},\mathsf{R2}\} : \mathsf{start}$, there must exist either a sequence of interactions ending in $\mathsf{R1} \to \mathsf{Ctrl} : \mathsf{finish}$ or in $\mathsf{R2} \to \mathsf{Ctrl} : \mathsf{finish}$.

These formulas can be expressed by the modal (μ-calculus) logic used by mCRL2. Furthermore, we exploited in recent work [16] how to generate, given a set of components of a team automaton and its synchronisation policies, both a mCRL2 model and a set of formulas that can guarantee progress. I.e., that any component who want to send a message can do so, and any component that is ready to receive messages can receive at least one of them. These two concepts are known in the literature as receptiveness and responsiveness [6], respectively.

3.2 Realisability challenges

Realising a global specification means inferring the local behaviour of each of the involved participants, under some assumptions regarding their communication

channels. We claim that realisations should satisfy the same properties of their original global models. Hence we should rely on bisimulation to compare their behaviour when using dynamic logic, since satisfaction of (propositional) dynamic logic formulas is invariant under bisimulation [19]. The choice of a different logic would lead to different equivalence notions.

Realisability has been extensively studied in the context of MPST, often guided by strict syntactic restrictions over the global protocol. These restrictions facilitate the process of building the local behaviour of each participant, by projecting the interactions to each of these participants, and provide computationally simpler mechanisms to guarantee correctness of the realisation.

Realisability has also been studied for CA [4] and for TA [13]; in both cases addressing language equivalence rather than bisimulation equivalence when comparing behaviours. In CA [4] building a local behaviour for a given participant P means producing an automaton that uses only interactions in which P is involved, such that the language is the same as the language of the global model restricted to these interactions. For instance, it is enough to hide interactions in which P does not appear (replacing them by τ), and minimise the automata collapsing these transitions. In TA [13] the language of a TA is characterised by the so-called synchronised shuffling of the behaviour of its components. Doing so, however, does not guarantee in neither cases that the realisation will satisfy the same (dynamic) logical formulas as the global protocol. As a matter of fact, some formulas in dynamic logic may satisfy either the global protocol or the composed system (exclusively) if these are language equivalent but not bisimilar.

This leads us to our ongoing effort to calculate a realisation from a global behaviour [12], i.e., from the semantics of TA with interactions as labels. Unlike the work on MPST, we try to avoid imposing syntactic restrictions on our global model, allowing our starting point to be any transition system labelled by interactions with multiple participants, instead of targeting a more practical subclass of global models. And unlike the work on CA, we try to guarantee that the realisations are bisimilar to the original model.

We identify a set of challenges when reasoning over realisability of TA.

- **How rich are the local labels?** Our initial motivating example in Fig. 1 uses labels consisting of a message name and a direction (input or output). In constraint automata (Fig. 2) there is no direction (less information), and in both CA and MPST the local participants use in each label the name, the direction, and the other participants involved (more information). Hence there will be global specifications that can be realised when producing participants with richer labels, but that cannot be realised when some information is lost (such as all participants involved). Conversely, having less information in the labels enables more compact local participants, e.g., our controller does not need to distinguish the finish from any of the two runners, while the controller derived from MPST needs to consider any interleaving of these two.
- **How to specify global specifications?** Building an automaton labelled by interactions can easily become too large due to all the combinations of concurrent actions. Hence a more compact model, such as a calculus for

choreographies (with multiple senders and receivers) with a parallel operator, or other useful operators, seems to be preferable. Alternatively models based on event structures [1] or Petri Nets could also provide a compact representation of concurrent actions. Regarding the latter, the *Vector Team Automata* variation of TA [14], where vectors of local labels restrict which components participate in each global label, has a composed semantics given by a form of labelled Petri Net called *Individual Token Net Controllers.*

– **Active learning?** Active learning approaches [25] try to infer the behaviour of a system by observing the actions of an input-deterministic black-box, assuming some mechanism to discover that the inferred model is good enough. Hence, a technique to infer the behaviour of local agents from traversing a (potentially large) global state could also be adapted to infer the behaviour of a set of agents from observing and reacting to ongoing interactions. This would be an alternative to produce the global state from a given model.

– **Other communication channels?** TA focus on synchronous interactions. Relaxing this to asynchronous interactions with many participants would largely increase the complexity of the realisability analysis. Furthermore, many variations are considered in the literature, usually fixed upfront. E.g., assuming a single sorted queue between each pair of participants (blocking if the first message cannot be processed), assuming there is no order on messages, assuming there is an order that gives priority to earlier messages (but allows skipping messages), and so on. Better understanding the impact of these, or even supporting the combination of these channel mechanisms, could improve the scope of applicability of existing tools and analysis.

– **How to model families of global specifications?** Variability has been studied in TA [8] and in other models such as BIP [27], Reo [34], and Petri Nets [32]. Variability in TA, BIP, and Petri Nets meant annotating transitions with conditions over a set of features that describe whether they should be included in a given configuration. However, these are not meant to describe, e.g., a family of systems with a number n of runners, for any $n > 0$. This was attempted with Reo [34], but using a complex calculus and not targetting automatic analysis. Hence finding a good model to describe variability on the number of participants, and exploit it in the analysis of TA or providing tool support, could be a good fit for TA, which already describes desired numbers of participants involved in each channel.

4   Conclusions

This paper revisits the constrained multiparty synchronisation present in Team Automata, relating it to other popular concurrency models, guided by a race example. It further addresses verification of TA via dynamic logics, and provides a direction and challenges on how to realise team automata from global specifications. By avoiding technical details and following an example-first approach, we expect this paper to be a nice introduction to concurrency models that can synchronise multiple participants, and to provide inspiration on topics and directions that we find relevant in this area.

Acknowledgments

References

1. Arbach, Y., Karcher, D.S., Peters, K., Nestmann, U.: Dynamic causality in event structures. Log. Methods Comput. Sci. **14**(1) (2018). https://doi.org/10.23638/LMCS-14(1:17)2018
2. Baier, C., Sirjani, M., Arbab, F., Rutten, J.J.M.M.: Modeling component connectors in Reo by constraint automata. Sci. Comput. Program. **61**(2), 75–113 (2006). https://doi.org/10.1016/j.scico.2005.10.008
3. Barbanera, F., Lanese, I., Tuosto, E.: Choreography Automata. In: COORDINATION. LNCS, vol. 12134, pp. 86–106. Springer (2020). https://doi.org/10.1007/978-3-030-50029-0_6
4. Barbanera, F., Lanese, I., Tuosto, E.: Formal Choreographic Languages. In: COORDINATION. LNCS, vol. 13271, pp. 121–139. Springer (2022). https://doi.org/10.1007/978-3-031-08143-9_8
5. ter Beek, M.H.: Team Automata: A Formal Approach to the Modeling of Collaboration Between System Components. Ph.D. thesis, Leiden University (2003)
6. ter Beek, M.H., Carmona, J., Hennicker, R., Kleijn, J.: Communication Requirements for Team Automata. In: COORDINATION. LNCS, vol. 10319, pp. 256–277. Springer (2017). https://doi.org/10.1007/978-3-319-59746-1_14
7. ter Beek, M.H., Carmona, J., Kleijn, J.: Conditions for Compatibility of Components: The Case of Masters and Slaves. In: ISoLA. LNCS, vol. 9952, pp. 784–805. Springer (2016). https://doi.org/10.1007/978-3-319-47166-2_55
8. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Featured Team Automata. In: FM. LNCS, vol. 13047, pp. 483–502. Springer (2021). https://doi.org/10.1007/978-3-030-90870-6_26
9. ter Beek, M.H., Ellis, C.A., Kleijn, J., Rozenberg, G.: Synchronizations in Team Automata for Groupware Systems. Comput. Sup. Coop. Work **12**(1), 21–69 (2003). https://doi.org/10.1023/A:1022407907596
10. ter Beek, M.H., Gadducci, F., Janssens, D.: A calculus for team automata. ENTCS **195**, 41–55 (2008). https://doi.org/10.1016/j.entcs.2007.08.022
11. ter Beek, M.H., Hennicker, R., Kleijn, J.: Compositionality of Safe Communication in Systems of Team Automata. In: ICTAC. LNCS, vol. 12545, pp. 200–220. Springer (2020). https://doi.org/10.1007/978-3-030-64276-1_11
12. ter Beek, M.H., Hennicker, R., Proença, J.: Realisability of global models of interaction. In: ICTAC. LNCS, Springer (2023)
13. ter Beek, M.H., Kleijn, J.: Team Automata Satisfying Compositionality. In: FME. LNCS, vol. 2805, pp. 381–400. Springer (2003). https://doi.org/10.1007/978-3-540-45236-2_22

14. ter Beek, M.H., Kleijn, J.: Vector Team Automata. Theor. Comput. Sci. **429**, 21–29 (2012). https://doi.org/10.1016/j.tcs.2011.12.020

15. ter Beek, M.H., Lenzini, G., Petrocchi, M.: Team Automata for Security: A Survey. Electron. Notes Theor. Comput. Sci. **128**(5), 105–119 (2005). https://doi.org/10.1016/j.entcs.2004.11.044

16. ter Beek, M.H., Cledou, G., Hennicker, R., Proença, J.: Can we communicate? using dynamic logic to verify team automata. In: Chechik, M., Katoen, J.P., Leucker, M. (eds.) Proceedings of the 25th International Symposium on Formal Methods (FM 2023). Lecture Notes in Computer Science, vol. 14000. Springer (2023). https://doi.org/10.1007/978-3-031-27481-7_9

17. ter Beek, M.H., Lenzini, G., Petrocchi, M.: A team automaton scenario for the analysis of security properties of communication protocols. J. Autom. Lang. Comb. **11**(4), 345–374 (2006). https://doi.org/10.25596/jalc-2006-345, https://doi.org/10.25596/jalc-2006-345

18. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures. Lecture Notes in Computer Science, vol. 3185, pp. 200–236. Springer (2004). https://doi.org/10.1007/978-3-540-30080-9_7

19. van Benthem, J., van Eijck, J., Stebletsova, V.: Modal Logic, Transition Systems and Processes. J. Log. Comput. **4**(5), 811–855 (1994). https://doi.org/10.1093/logcom/4.5.811

20. Bliudze, S., Sifakis, J.: The algebra of connectors - structuring interaction in BIP. IEEE Trans. Computers **57**(10), 1315–1330 (2008). https://doi.org/10.1109/TC.2008.26

21. Bunte, O., Groote, J.F., Keiren, J.J.A., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.C.: The mCRL2 Toolset for Analysing Concurrent Systems. In: TACAS. LNCS, vol. 11428, pp. 21–39. Springer (2019). https://doi.org/10.1007/978-3-030-17465-1_2

22. Carmona, J., Kleijn, J.: Compatibility in a multi-component environment. Theor. Comput. Sci. **484**, 1–15 (2013). https://doi.org/10.1016/j.tcs.2013.03.006

23. de Alfaro, L., Henzinger, T.A.: Interface Automata. In: ESEC/FSE. pp. 109–120. ACM (2001). https://doi.org/10.1145/503209.503226

24. Dokter, K., Jongmans, S., Arbab, F., Bliudze, S.: Combine and conquer: Relating BIP and Reo. J. Log. Algebraic Methods Program. **86**(1), 134–156 (2017). https://doi.org/10.1016/j.jlamp.2016.09.008

25. al Duhaiby, O., Groote, J.F.: Active learning of decomposable systems. In: Bae, K., Bianculli, D., Gnesi, S., Plat, N. (eds.) FormaliSE@ICSE 2020: 8th International Conference on Formal Methods in Software Engineering, Seoul, Republic of Korea, July 13, 2020. pp. 1–10. ACM (2020). https://doi.org/10.1145/3372020.3391560

26. Ellis, C.A.: Team Automata for Groupware Systems. In: Proceedings of the 1st International ACM SIGGROUP Conference on Supporting Group Work (GROUP). pp. 415–424. ACM (1997). https://doi.org/10.1145/266838.267363

27. Farhat, S., Bliudze, S., Duchien, L., Kouchnarenko, O.: Toward run-time coordination of reconfiguration requests in cloud computing systems. In: Jongmans, S., Lopes, A. (eds.) Coordination Models and Languages - 25th IFIP WG 6.1 International Conference, COORDINATION 2023, Held as Part of the 18th International Federated Conference on Distributed Computing Techniques, DisCoTec 2023, Lisbon, Portugal, June 19-23, 2023, Proceedings. Lecture Notes in Computer

Science, vol. 13908, pp. 271–291. Springer (2023). https://doi.org/10.1007/978-3-031-35361-1_15

28. Ghilezan, S., Jaksic, S., Pantovic, J., Scalas, A., Yoshida, N.: Precise subtyping for synchronous multiparty sessions. J. Log. Algebraic Methods Program. **104**, 127–173 (2019). https://doi.org/10.1016/j.jlamp.2018.12.002

29. Jongmans, S.T.Q., Arbab, F.: Overview of thirty semantic formalisms for Reo. Sci. Ann. Comput. Sci. **22**(1), 201–251 (2012). https://doi.org/10.7561/SACS.2012.1.201

30. Kokash, N., Krause, C., de Vink, E.P.: Reo + mCRL2: A framework for model-checking dataflow in service compositions. Formal Aspects Comput. **24**(2), 187–216 (2012). https://doi.org/10.1007/s00165-011-0191-6

31. Lynch, N.A., Tuttle, M.R.: An Introduction to Input/Output Automata. CWI Q. **2**(3), 219–246 (1989), https://ir.cwi.nl/pub/18164

32. Muschevici, R., Proença, J., Clarke, D.: Feature Nets: behavioural modelling of software product lines. Softw. Sys. Model. **15**(4), 1181–1206 (2016). https://doi.org/10.1007/s10270-015-0475-z

33. Orlando, S., Pasquale, V.D., Barbanera, F., Lanese, I., Tuosto, E.: Corinne, a tool for choreography automata. In: Salaün, G., Wijs, A. (eds.) Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13077, pp. 82–92. Springer (2021). https://doi.org/10.1007/978-3-030-90636-8_5

34. Proença, J., Clarke, D.: Typed connector families. In: Braga, C., Ölveczky, P.C. (eds.) Formal Aspects of Component Software - 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9539, pp. 294–311. Springer (2015). https://doi.org/10.1007/978-3-319-28934-2_16

35. Proença, J., Madeira, A.: Taming hierarchical connectors. In: Hojjat, H., Massink, M. (eds.) Fundamentals of Software Engineering - 8th International Conference, FSEN 2019, Tehran, Iran, May 1-3, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11761, pp. 186–193. Springer (2019). https://doi.org/10.1007/978-3-030-31517-7_13

36. Severi, P., Dezani-Ciancaglini, M.: Observational equivalence for multiparty sessions. Fundam. Informaticae **170**(1-3), 267–305 (2019). https://doi.org/10.3233/FI-2019-1863