Corinne, a Tool for Choreography Automata

# Corinne, a Tool for Choreography Automata*

Simone Orlando[1], Vairo Di Pasquale[1], Franco Barbanera[2][0000−0002−8039−1085], Ivan Lanese[3][0000−0003−2527−9995], and Emilio Tuosto[4][0000−0002−7032−3281]
simoneorlando.cs@gmail.com,vairo.dp@gmail.com,barba@dmi.unict.it,
ivan.lanese@gmail.com, emilio.tuosto@gssi.it

[1] University of Bologna (Italy)
[2] Dept. of Mathematics and Computer Science, University of Catania (Italy)
[3] Focus Team, University of Bologna/INRIA (Italy)
[4] Gran Sasso Science Institute (Italy)

**Abstract.** Choreography automata are a model of choreographies envisaging high-level views of the behaviour of communicating systems as finite-state automata. The behaviour of each participant of a choreography can be obtained via a projection operation from a choreography automaton. The system of participants obtained by projection is well-behaved if the choreography automaton satisfies some well-formedness conditions. We present Corinne, a tool allowing one to render, compute projections of and compose choreography automata, as well as to check well-formedness conditions.

## 1 Introduction

Programming and understanding distributed systems is notoriously difficult due to the need to reason on multiple flows of execution and many possible behaviours; yet distributed systems are fundamental nowadays. Indeed most of our systems, from social networks to apps, from games to scientific software, are distributed. A main challenge when programming distributed systems, in particular multiparty ones, is how to define communication protocols avoiding subtle bugs such as deadlocks.

In order to reason on the correctness and properties of multiparty communication protocols, dedicated models such as conversation protocols [24], choreographies [11,28,31], global graphs [35], and multiparty session types [12,26,27] have been proposed. Their common trait is to provide global descriptions of the behaviour of a distributed system, and to allow one to ensure desirable properties such as deadlock freedom by checking some structural conditions on the model. Also, they provide an operation, called projection, to extract from the global specification a description of the (local) behaviour that each participant has to follow in order to implement the desired global behaviour.

In this paper we focus on choreography automata (c-automata) [4], which are an automata model belonging to the family described above. Essentially, c-automata are finite-state automata whose transitions are labelled by interactions representing point-to-point communications between a sender and a receiver. Despite its simplicity, manually performing the constructions and analysis

* Research partly supported by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No 778233, by the MIUR project PRIN 2017FTXR7S "IT-MaTTerS" (Methods and Tools for Trustworthy Smart Systems), and by the Progetto di Ateneo Pia.Ce.Ri - UNICT. The third and fourth authors have also been partially supported by INdAM as members of GNCS (Gruppo Nazionale per il Calcolo Scientifico). The authors thank the reviewers for their interesting comments and suggestions, which helped us to improve the paper. The third author wishes to thank also Mariangiola Dezani-Ciancaglini for her support.

(such as the ones in [4]) on c-automata is tedious, error prone even for simple cases, and its complexity increases with the size of c-automata to the point that it becomes practically impossible even on moderately large instances.

Thus, we decided to automate the main constructions and analysis on c-automata in a prototype tool called Corinne. This tool allows us to experiment with c-automata. We illustrate the usefulness of Corinne by applying it to the examples in [4,3]. This exercise allowed us to spot a couple of (minor) errors in [4]. We will come back to this when describing the tool. It is worth noticing that the definitions of choreography automaton and of its projection are independent of the chosen communication model (synchronous or asynchronous). Indeed, this choice affects only the definition of well-formedness conditions.

After reviewing the main constructions and operations of c-automata (§ 2), we introduce Corinne (§ 3). We will conclude the paper (§ 4) with some final remarks.

Corinne is available at [14] (under the open-source MIT license), together with all the examples discussed in this paper.

## 2 Choreography Automata

This section surveys choreography automata (c-automata) borrowing definitions and concepts from [4,5]; for full details about this formalism we refer the reader to [4,5]. C-automata (ranged over by $\mathbb{C}$A, $\mathbb{C}$B, etc.) are Finite-State Automata (FSAs) whose transitions are labelled by interactions of the form A→B: m; such interaction represents a communication between participants A and B where the former sends a message (of type) m to the latter, which is supposed to receive m. We let $\lambda$ range over the set $\mathcal{L}_{\text{int}}$ of interactions.

**Definition 2.1 (Choreography automata).** *A* choreography automaton *(c-automaton) is an FSA on the alphabet $\mathcal{L}_{int}$, namely a tuple $\langle Q, q_0, \mathcal{L}_{int}, \rightarrow\rangle$ where $Q$ is a finite set of states, $q_0$ the initial state, and $\rightarrow \subseteq Q \times \mathcal{L}_{int} \times Q$ the transition relation. We write $q \xrightarrow{\lambda} q'$ when $(q, \lambda, q') \in \rightarrow$.*

Given a c-automaton, the projection operation builds the corresponding communicating system consisting of the set of projections of the c-automaton on each participant. Each projection is an FSA as well, on the alphabet $\mathcal{L}_{\text{act}}$ of actions, which have the form A B!m, A B?m. The former denotes the action of sending message m from A to B, the latter the corresponding receiving action. Such FSAs are called Communicating Finite State Machines (CFSMs) [10]. Hereafter, $\mathcal{P}_{\mathbb{C}\text{A}}$ denotes the set of participants of a c-automaton $\mathbb{C}$A; note that $\mathcal{P}_{\mathbb{C}\text{A}}$ is necessarily finite.

**Definition 2.2 (Automata projection).** *The* projection on A *of a transition $t = q \xrightarrow{\lambda} q'$ of a c-automaton, written $t{\downarrow_{\text{A}}}$, is defined by:*

$$t{\downarrow_{\text{A}}} = \begin{cases} q \xrightarrow{\text{A B!m}} q' & \text{if } \lambda = \text{A}{\rightarrow}\text{B}\colon \text{m} \\ q \xrightarrow{\text{B A?m}} q' & \text{if } \lambda = \text{B}{\rightarrow}\text{A}\colon \text{m} \\ q \xrightarrow{\epsilon} q' & \text{otherwise} \end{cases}$$

*The* projection *of a c-automaton $\mathbb{C}\text{A} = \langle Q, q_0, \mathcal{L}_{int}, \rightarrow\rangle$ on a participant $\text{A} \in \mathcal{P}_{\mathbb{C}\text{A}}$, denoted $\mathbb{C}\text{A}{\downarrow_{\text{A}}}$, is obtained by determinising*[1] *up-to-language equivalence the* intermediate *automaton*

$$A_{\text{A}} = \langle Q, q_0, \mathcal{L}_{act} \cup \{\,\epsilon\,\}, \{\,(q \xrightarrow{\lambda} q'){\downarrow_{\text{A}}} \mid q \xrightarrow{\lambda} q'\,\}\rangle$$

*The* projection of $\mathbb{C}$A, *written $\mathbb{C}\text{A}{\downarrow}$, is the communicating system $(\mathbb{C}\text{A}{\downarrow_{\text{A}}})_{\text{A}\in\mathcal{P}_{\mathbb{C}\text{A}}}$.*

[1] In [4] also minimisation is performed, but this is not needed for the correctness of the constructions, and it is not currently performed by Corinne.

The projection of c-automata is essentially obtained by transferring projections of global specifications present in several choreography-based approaches such as, e.g., [12,26,13,27,35]. A composition operation on c-automata has been recently proposed by the last three authors in [5]. The idea is to *lift* at the choreographic level a version of the composition of systems of CFSMs described in [2] and applied in a multiparty session type setting in [3]. This technique enables to overcome the fact that in choreographic approaches systems are usually intended to be *closed*. Actually, it is instead possible to look at any system as an *open* one (so enabling modular development) by looking at any of its participants as a possible interface. Hence, the composition of systems is essentially obtained by taking two systems, selecting two of their participants (one per system) provided that they meet some *compatibility* conditions, and removing them while redirecting communications to them towards the other system. More precisely, if a message is sent by some participant A to the chosen interface of the system it belongs to, compatibility conditions require the interface of the other system to send an identical message to some participant B. In the composed system A sends the message directly to B. This way of composing systems can be obtained by applying one after the other two operations:

1. the product of c-automata, building a c-automaton corresponding to the concurrent execution of the two original c-automata; and
2. a *blending* operation that, given two participants (the chosen interfaces) of a same c-automaton, removes them and adjusts the c-automaton as described above.

The formalisation of these operations can be found in [5], while an example will be discussed in the next section (Fig. 3).

## 3 Corinne

The operations of projection and composition of c-automata described in § 2 are implemented in Corinne [14]. The tool is written in python3 and works on c-automata represented as particular directed graphs in the DOT format [23]. Rendering of DOT files is performed using the graphviz library [25]. Other formats can be used as input of Corinne; more precisely the tool also parses regular expressions used as the syntax of global graphs [35] in ChorGram [15,17], or the DOT representation [23] of global graphs produced by Domitilla [22]. We remark that only global graphs with no parallel composition correspond to c-automata and can thus be imported. All parsers are defined using ANTLR4 [32].

Users interact with Corinne through a graphical interface based on the tkinter package [34]. The GUI of Corinne displays FSAs that are either c-automata or CFSMs obtained via projection. As shown in the screenshot in Fig. 1, each FSA appears in a separate tab. The tab also reports basic information on the FSA (e.g., number of states and of edges) as well as a graphical rendering of the FSA itself.

Besides utility menus `File` and `Help`, Corinne has two menus to work on c-automata. Menu `Transformations` allows one to compute projections[2] on a given participant, the product of two c-automata as well as the blending (synchronisation) operation via interfaces following the approach described in [5].

Menu `Properties` instead allows one to check the well-formedness conditions discussed in [4] ensuring that the language of the c-automaton coincides with the one of the (synchronous) system obtained via projection, and that the latter is live, lock-free, and deadlock-free (we refer to [4] for

[2] Determinisation required for projection is computed using the classical subset construction for FSAs with $\epsilon$-transitions.

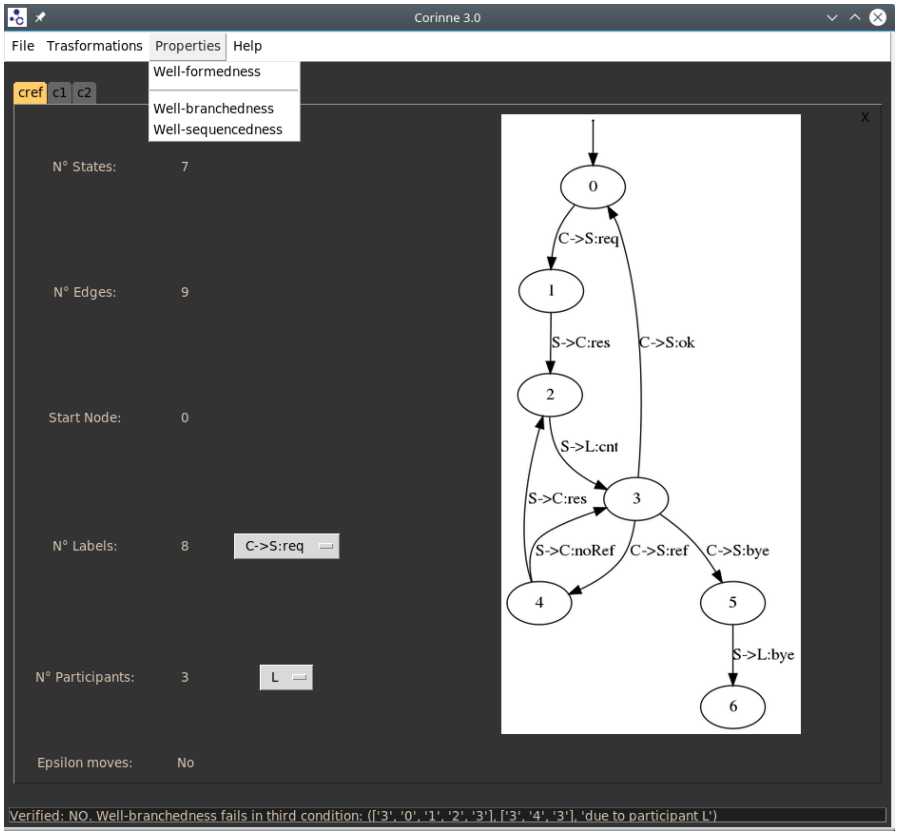


**Fig. 1.** Corinne screenshot

the definition of these properties in the context of c-automata as well as for formal statements and proofs of the results hinted at above).

The screenshot in Fig. 1 depicts the c-automaton ℂref used in [4, Introduction] as a running example. ℂref specifies the coordination among participants C, S, and L whereby a request req from client C is served by server S which replies with a message (of type) res and logs some meta-information cnt on a service L (e.g., for billing purposes). Client C may acknowledge a response of S (*i*) with an ok message to restart the protocol, or (*ii*) by requiring a refinement of the response with a ref message, or else (*iii*) by ending the protocol with a bye message which S forwards to L. In the second case, S sends C either a noRef message, if no refinement is possible, or another res (with the corresponding cnt to L). Using Corinne we can generate the projections of ℂref. Fig. 2 contrasts the projection on participant S returned by Corinne and the one in [4, Example 3.4], manually computed. The two CFSMs differ on the labels of the transition from state 4 to 6 and {5} to {6}, respectively from the left and the right CFSM, which should correspond to each other. In fact, the label S C!bye on the transition from {5} to {6} is wrong.

We can also check the well-formedness of ℂref, which is the conjunction of two conditions, well-branchedness and well-sequencedness. Intuitively, well-branchedness requires that all the participants are aware of which branch is taken in a choice, if they have to behave differently on the available branches. Well-sequencedness instead requires concurrency (due to communications

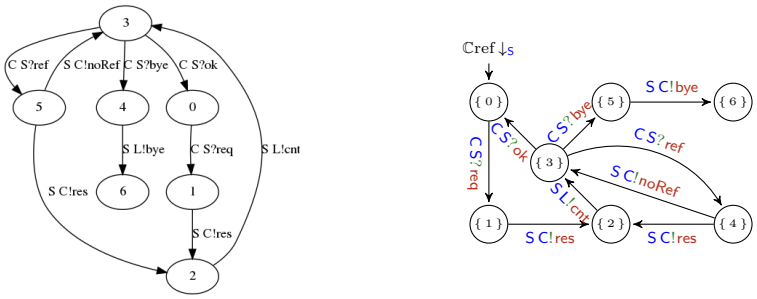


**Fig. 2.** Projections of ℂref on S from Corinne (left) and [4] (right)

involving disjoint sets of participants) to be explicitly represented as commuting diamonds. As expected Corinne reports ℂref to be well-sequenced. Unexpectedly, the check of well-branchedness fails. This is shown in the message in the bottom part of Fig. 1. The message means that the third condition in [4, Definition 4.6] fails on the pair of paths `3-0-1-2-3` and `3-4-3` because of participant L. The reason is that L occurs in the former but not in the latter. This implies that, in case the system reaches state 3 and participant C keeps on choosing indefinitely to send message ref to S, participant L will never be aware of what is going on. So L gets stuck waiting for a message bye that will never arrive. We refer to [4] for further details on well-formedness conditions. We conjecture that ℂref is nevertheless well-behaved under suitable fairness assumptions, but the theory in [4] needs to be generalised to prove it.

We now demonstrate the composition operation relying on the running example of [3], where (referring to the UML representations) the diagrams in [3, Fig. 5] and [3, Fig. 6] are composed to derive the one in [3, Fig. 8]. Fig. 3 shows the two c-automata involved and the result of the composition. The top-left c-automaton (let's dub it ℂA1) represents the global behaviour of a system with participants P, Q, and H interacting according to the following protocol. Participant P keeps on sending text messages to Q, which has to deliver them to H. Participant P can send a new message only if H has ascertained the propriety of language of the previous one, i.e if the latter does not contain, say, rude or offensive words. Participant H acknowledges to Q the propriety of language of a received text by means of the message ack. In such a case, Q sends to P an ok message so that P can proceed by sending a further message. If the message does not passes the check, then H sends a nack message to inform Q that the text has not the required propriety of language. In such a case, Q produces transf (a semantically invariant reformulation of the text), sends it back to H and so that it can be checked. Before doing that, Q informs P (through the notyet message) that the text has not been accepted yet and a reformulation has been requested. After receiving a message, H may also decide to stop the interaction, sending a stop message to inform Q that no more text will be accepted. In such a case, Q informs P of that.

The bottom-left c-automaton (let's dub it ℂA2), instead describes a system formed by participants K, R, and S interacting according to the following protocol. Participant K sends text messages to R and S in an alternating way, starting with R. Participants R and S inform K that a text has been accepted or refused by sending back, respectively, either ack or nack. In the former case, it is the other receiver's turn to receive the text: a message go is exchanged between R and S to signal this case. In case nack is sent back, the sender has to resend the text until it is accepted. Meanwhile, the participant currently selected by K asks the other one to wait, since the previous message is being resent in a transformed form.

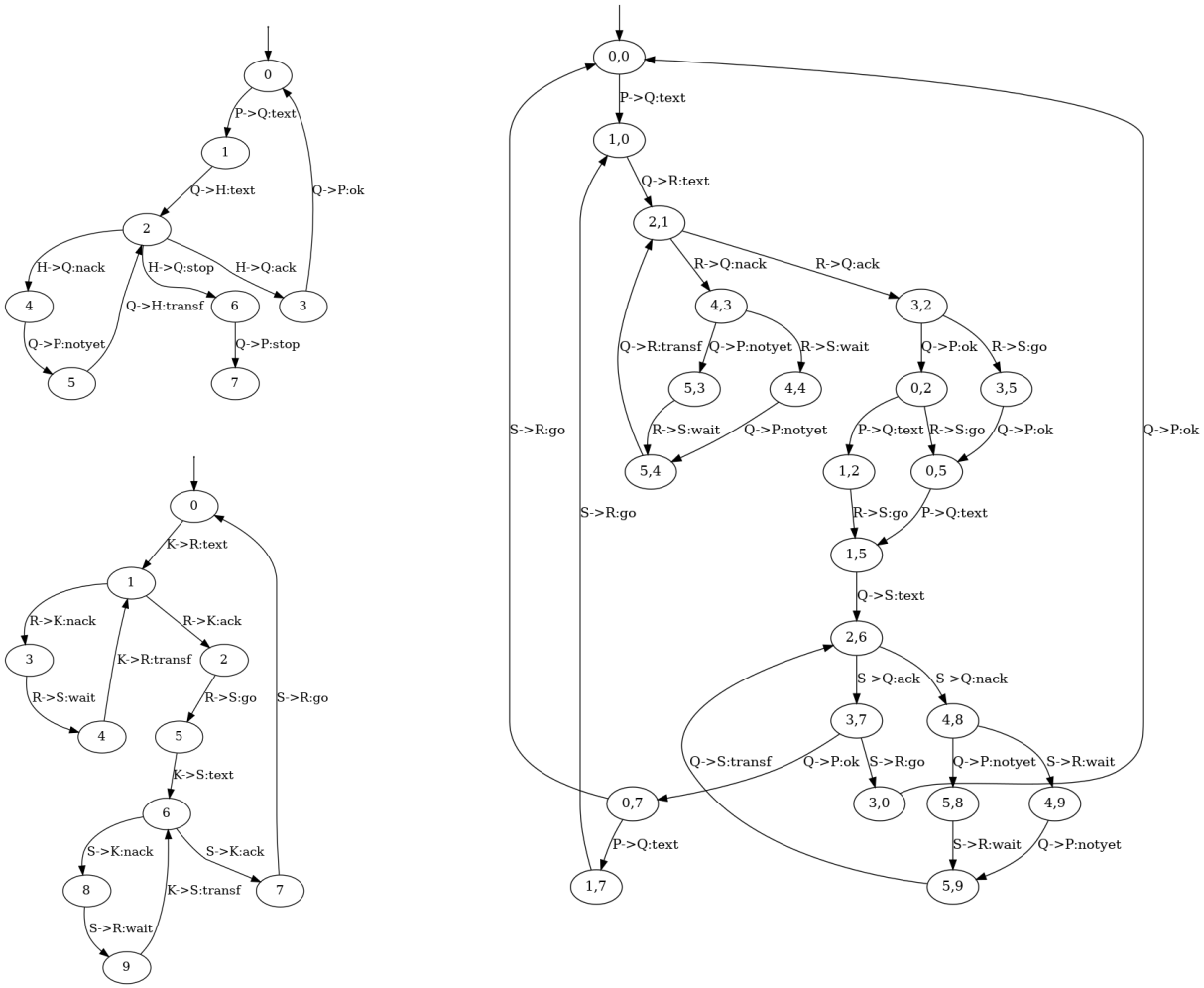


**Fig. 3.** Composition of c-automata in Corinne

In the composition, participants H and K in, respectively, $\mathbb{C}$A1 and $\mathbb{C}$A2 of Fig. 3 are chosen as interfaces. This means that, e.g., when participant Q sends a text, it will send it alternatively to R and S. This can be thought somehow as if $\mathbb{C}$A1 invokes $\mathbb{C}$A2 for sending the message. However, w.r.t. choreographies with procedure invocations such as [20,19], our approach on the one hand allows a complex interaction between caller and callee choreographies but, on the other hand, does not allow for parameter passing in the invocation. Notice that the interfaces H and K are *compatible*; specifically, the languages of $\mathbb{C}$A1$\downarrow_{\mathsf{H}}$ and $\mathbb{C}$A2$\downarrow_{\mathsf{K}}$ are *dual* to each other if we disregard the name of participants other than H and K in the input/output actions (duality corresponds to the exchange of '!' with '?' and vice versa in the actions). Compatibility, roughly, enables the composition not to modify in an essential way the behaviour of participants other than H and K. To obtain in Corinne the composition of $\mathbb{C}$A1 and $\mathbb{C}$A2 via the chosen interfaces H and K, we need first to apply `Product` on the two c-automata and then to apply `Synch` on H and K on the result. The only relevant difference between the composition performed by Corinne and the one in [3] is that the synchronisation in [3] transforms H and K into gateways while our composition

drops them. Actually, our composition precisely corresponds to the direct composition in [3], but there is no example in [3] of this form of composition. We can obtain our result from the one in [3] by transforming sequences of communications `A-H-K-B` into `A-B` and `B-K-H-A` into `B-A` for any `A` and `B`. We also remark that the representation as c-automata highlights concurrency as commuting diamonds, e.g., the one at states `(4,3),(5,3),(4,4),(5,4)`. Both the component c-automata and their composition can be checked to be both well-sequenced and well-branched. However, the check of well-branchedness on the composition is quite heavy.

## 4 Conclusion, Related Work, and Future Work

We refer to [4] for a comparison between c-automata and related models, while here we focus on the relations between Corinne and the most related tools.

Possibly Corinne's closest sibling is ChorGram, a tool chain based on global graphs to support choreographic development of message-oriented applications [17,29]. Global graphs are not directly comparable with c-automata: on the one hand they are more general since they allow one to specify parallelism, but on the other hand they require structured interactions. As a result, global graphs without parallel composition correspond to a strict subset of c-automata. In a sense, ChorGram complements Corinne's functionalities; for instance, it supports different semantics of global graphs and some experimental ideas on choreography amendment or model-driven testing of message passing applications [18]. While Corinne can already take as input global graphs without parallel composition produced by ChorGram, we plan to further integrate the two tools in the future. We are currently considering to encode the parallel composition of global graphs as interleaving of independent transitions. We also plan to extend ChorGram so that it imports c-automata produced by Corinne. This paves the way to extensions of Corinne with features to import models based, e.g., on multiparty session types such as [33,21] once proper mappings to c-automata are defined. We remark that this might not be simple for models relying on asynchronous communications such as [30,13] for Corinne's semantics is synchronous.

Another toolkit close to Corinne is CAT, a tool introduced in [6] to support the verification of communication protocols expressed as contract automata via the analysis of *agreement* properties. Contract automata are a versatile model of automata featuring the synthesis of controllers for communicating components; a thorough analysis based on CAT of the relations between choreographic- and orchestration-based controllers (initiated in [7]) has been recently developed in [9]. This suggests a possible entanglement of the complementary features of Corinne and CAT also in the light of the recent refactoring of the latter tool [8]. In fact, recently this model is being applied to choreography automata; Corinne could be useful in this context to validate choreography automata synthesised with contract automata.

We believe that Corinne is useful to experiment with c-automata, yet a number of improvements are desirable. First, right now the complexity of the check for well-branchedness is too high. We believe this can be reduced, at least in the average case, by avoiding checking multiple times analogous choices which are repeated in many states due to concurrency. Also, other functionalities would be useful, such as performing composition via gateways as described in [3] or checking well-formedness conditions also for the asynchronous semantics [4].

## References

1. Franco Barbanera, Ugo de'Liguoro, and Rolf Hennicker. Global types for open systems. In Massimo Bartoletti and Sophia Knight, editors, *ICE*, volume 279 of *EPTCS*, pages 4–20, 2018.

2. Franco Barbanera, Ugo de'Liguoro, and Rolf Hennicker. Connecting open systems of communicating finite state machines. *Journal of Logic and Algebraic Methods in Programming*, 109, 2019. Extended version of [1].
3. Franco Barbanera, Mariangiola Dezani-Ciancaglini, Ivan Lanese, and Emilio Tuosto. Composition and decomposition of multiparty sessions. *Journal of Logic and Algebraic Methods in Programming*, 119:100620, 2021.
4. Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Choreography automata. In Simon Bliudze and Laura Bocchi, editors, *COORDINATION*, volume 12134 of *LNCS*, pages 86–106. Springer, 2020.
5. Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Composition of choreography automata. Technical Report 2107.06727, Arxiv, July 2021. `http://arxiv.org/abs/2107.06727`.
6. Davide Basile, Pierpaolo Degano, GianLuigi Ferrari, and Emilio Tuosto. Playing with our CAT and communication-centric applications. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Networked and Distributed Systems*, volume 9688 of *LNCS*, pages 62–73. Springer, 2016.
7. Davide Basile, Pierpaolo Degano, GianLuigi Ferrari, and Emilio Tuosto. Relating two automata-based models of orchestration and choreography. *Journal of Logic and Algebraic Methods in Programming*, 85(3):425–446, 2016.
8. Davide Basile and Maurice H. ter Beek. A clean and efficient implementation of choreography synthesis for behavioural contracts. In Ferruccio Damiani and Ornela Dardha, editors, *COORDINATION*, volume 12717 of *LNCS*, pages 225–238. Springer, 2021.
9. Davide Basile, Maurice H. ter Beek, and Rosario Pugliese. Synthesis of orchestrations and choreographies: Bridging the gap between supervisory control and coordination of services. *Logical Methods in Computer Science*, 16(2), 2020.
10. Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.
11. Mario Bravetti and Gianluigi Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In Markus Lumpe and Wim Vanderperren, editors, *Software Composition*, volume 4829 of *LNCS*, pages 34–50. Springer, 2007.
12. Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centred programming for web services. In Rocco De Nicola, editor, *ESOP*, volume 4421 of *LNCS*, pages 2–17. Springer, 2007.
13. Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *POPL*, pages 263–274, 2013.
14. Corinne github repository. `https://github.com/lanese/corinne-3`.
15. Alex Coto, Roberto Guanciale, Julien Lange, and Emilio Tuosto. ChorGram: tool support for choreographic development. Available at `https://bitbucket.org/emlio_tuosto/chorgram/wiki/Home`, 2015.
16. Alex Coto, Roberto Guanciale, and Emilio Tuosto. An abstract framework for choreographic testing. In Julien Lange, Anastasia Mavridou, Larisa Safina, and Alceste Scalas, editors, *Proceedings 13th Interaction and Concurrency Experience, ICE 2020, Online, 19 June 2020*, volume 324 of *EPTCS*, pages 43–60, 2020.
17. Alex Coto, Roberto Guanciale, and Emilio Tuosto. Choreographic development of message-passing applications - A tutorial. In Simon Bliudze and Laura Bocchi, editors, *COORDINATION*, volume 12134 of *LNCS*, pages 20–36. Springer, 2020.
18. Alex Coto, Roberto Guanciale, and Emilio Tuosto. An abstract framework for choreographic testing. *Journal of Logic and Algebraic Methods in Programming*, 123:100712, 2021. Extended version of [16].
19. Luís Cruz-Filipe and Fabrizio Montesi. Procedural choreographic programming. In *FORTE*, volume 10321 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2017.
20. Romain Demangeon and Kohei Honda. Nested protocols in session types. In *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2012.
21. Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. In Simon Gay and Jade Alglave, editors, *Proceedings Eighth International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES 2015, London, UK, 18th April 2015*, volume 203 of *EPTCS*, pages 29–43, 2015.

22. Domitilla github repository. https://github.com/dedo94/Domitilla.
23. The DOT Language. https://graphviz.org/doc/info/lang.html.
24. Xiang Fu, Tevfik Bultan, and Jianwen Su. Conversation protocols: a formalism for specification and verification of reactive electronic services. *Theor. Comput. Sci.*, 328(1-2):19–37, 2004.
25. Graphviz 0.16 - Simple Python interface for Graphviz. https://pypi.org/project/graphviz/.
26. Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *POPL*, pages 273–284. ACM Press, 2008.
27. Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.
28. Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, Yves Lafon, and Charlton Barreto. Web services choreography description language version 1.0. Technical report, W3C, 2005. http://www.w3.org/TR/ws-cdl-10/.
29. Julien Lange, Emilio Tuosto, and Nobuko Yoshida. A tool for choreography-based analysis of message-passing software. In Simon Gay and Antonio Ravara, editors, *Behavioural Types: from Theory to Tools*, Automation, Control and Robotics, chapter 6, pages 125–146. River, 2017.
30. Nicholas Ng, Nobuko Yoshida, and Kohei Honda. Multiparty session C: safe parallel programming with message optimisation. In Carlo A. Furia and Sebastian Nanz, editors, *Objects, Models, Components, Patterns - 50th International Conference, TOOLS 2012, Prague, Czech Republic, May 29-31, 2012. Proceedings*, volume 7304 of *Lecture Notes in Computer Science*, pages 202–218. Springer, 2012.
31. OMG. *Business Process Model and Notation (BPMN), Version 2.0*, January 2011. https://www.omg.org/spec/BPMN.
32. Terence Parr. Antlr. https://www.antlr.org/index.html.
33. Paula Severi and Mariangiola Dezani-Ciancaglini. Observational equivalence for multiparty sessions. *Fundam. Informaticae*, 170(1-3):267–305, 2019.
34. TKinter — Python interface to Tcl/Tk. https://docs.python.org/3/library/tkinter.html.
35. Emilio Tuosto and Roberto Guanciale. Semantics of global view of choreographies. *Journal of Logic and Algebraic Methods in Programming*, 95:17–40, 2018.