A Decentralized Analysis of Multiparty Protocols

Bas van den Heuvel and Jorge A. Pérez

University of Groningen, The Netherlands

This extended abstract summarizes a new approach to the analysis of the *protocols* that are essential in concurrent and distributed software. These protocols ensure a correct communication between independent programs. The important but difficult challenge is ensuring that communicating programs never violate the protocol and never get stuck in a deadlock. The work we describe here is currently under submission [21].

In this work, we focus on Multiparty Session Types (MPST) [12], in which protocols describe multiparty interactions from a vantage point. Consider the following motivating example, a *recursive authorization protocol*, involving three participants: a Client, a Server, and an Authorization service. The protocol specifies that the Server requests the Client to login, upon which the Authorization service mediates in the verification of the Client's password.

Multiparty Session Types In MPST, protocols are formally specified as *global types*. The following global type defines the just-described authorization protocol, with participants Client ('c'), Server ('s'), and Authorization service ('a'):

$$G_{\mathsf{auth}} = \mathsf{rec}\, X \,.\, s \twoheadrightarrow c \left\{ \begin{array}{l} \mathsf{login} \,.\, c \twoheadrightarrow a\{\mathsf{passwd}\langle \mathsf{str}\rangle \,.\, a \twoheadrightarrow s\{\mathsf{auth}\langle \mathsf{bool}\rangle \,.\, X\}\}, \\ \mathsf{quit} \,.\, c \twoheadrightarrow a\{\mathsf{quit} \,.\, \mathsf{end}\} \end{array} \right\}$$

The global type G_{auth} specifies a recursive protocol on the recursion variable X ('$\mathsf{rec}\, X$'). Then, the global type stipulates a message from s to c ('$s \twoheadrightarrow c$'), where s can choose between sending the label login or the label quit. The rest of the protocol depends on this initial choice by s. In the case of login, c sends to a the label passwd, along with a string value ('$\langle \mathsf{str}\rangle$'). Then, a sends to s the label auth, along with a boolean value ('$\langle \mathsf{bool}\rangle$'). Finally, the protocol starts over with a call on the recursion variable X. If s initially chooses quit, c forwards the quit label to a, after which the protocol ends ('end').

The theory of MPST analyzes implementations of global types as message-passing processes. An implementation of a global type consists of a network of individual processes, each of which implements the role of one of the global type's participants. We call these individual processes the global type's *participant implementations*. The analysis of MPST theory involves verifying the correctness of these networks of participant implementations. In this context, correctness follows from three essential properties: *protocol fidelity* (processes interact as stipulated by the global type), *communication safety* (no errors or mismatches in messages), and *deadlock freedom* (processes never get stuck waiting for each other). Verifying these properties is a challenging problem, which is further complicated by two salient features in message-passing concurrency, motivated in the context of our running example protocol:

- *Delegation*: the Client asks another participant, such as a Password manager, to act on their behalf;
- *Interleaving*: a single process implements the roles of both the Server and the Authorization service.

MPST has been widely studied, from both foundational and applied angles [3, 6, 13, 18, 1, 2, 19, 7, 14, 15]. Most works derived from the original theory by Honda *et al.* [11] define

behavioral type systems for a π-calculus to ensure protocol fidelity and communication safety. In the absence of delegation and interleaving, deadlock freedom is easy because the analysis only concerns single-threaded processes. However, in their presence, deadlock freedom becomes much harder, addressed only by some works [3, 16, 8].

Coordination and Topology In our work, we take on the challenge of verifying the correctness and deadlock freedom of networks of participant implementations of global types, including support for delegation and interleaving. The particulars of networks of communicating processes are informed by the way in which their interactions are coordinated. There are two salient approaches [17], which we motivate by analogies by Van der Aalst [22]:

- *Orchestration*, where a coordinator process ensures that all participants follow the protocol as intended, like the conductor of an orchestra;
- *Choreography*, where the processes directly follow the protocol without external coordination, like dancers individually dancing together to perform a global scenario.

The analysis of MPST falls under the choreography-based approach: each process is informed about their role in an implementation directly from the global type, interacting directly with each other without the support of an external coordinator. The coordination of networks of processes also has consequences for the *topologies* that these networks can be in. Orchestrated coordination imposes a *centralized* network topology, because all participating processes must connect to an additional, central component. However, topologies in choreographed coordination can range from centralized to *decentralized*, because the involved processes can connect directly to each other.

Dancing Shoes: A Decentralized Implementation In our work, we analyze decentralized implementations of global types. In particular, we work with implementations in a π-calculus with asynchronous communication, and use a *binary* session type system to verify the correctness of these implementations leveraging on existing results. Each of a global type's participant implementations has one particular channel on which they can send and receive protocol messages. Given a global type, we project the global type onto each of its participants, resulting in a binary session type for type checking the behavior of each of the participant implementations.

Because the participant implementations each only have a single channel for communication, we cannot connect them directly to each other to form a decentralized process network. Instead, we leverage on the global type to generate *router processes* (simply *routers*), that enable the participant implementations to communicate directly with each other. We generate a router for each participant, such that it redirects the communications on the single channel of the participant's implementation over several channels, one for every other participant, according to the participant's role in the global type. The routers then serve to "wrap" each participant implementation, forming a *routed implementation*. The resulting routed implementations can then be connected directly to each other, such that together they correctly realize the protocol specified by the global type. The configurations of networks of connected routed implementations are arbitrary, subsuming both centralized and decentralized topologies.

As they are *synthesized* from global types, routers do not change the behavior of participant implementations. We can look at routers in the context of Van der Aalst's analogy between dancers and choreographies of communicating processes: the participant implementations are barefoot dancers, and the routers provide them with the appropriate *dancing shoes* to perform their dance. More concretely, Figure 1 (left) illustrates a choreography of the routed implementations of the participants of G_{auth}: once wrapped in an appropriate router, the participant implementations can be connected directly to each other.

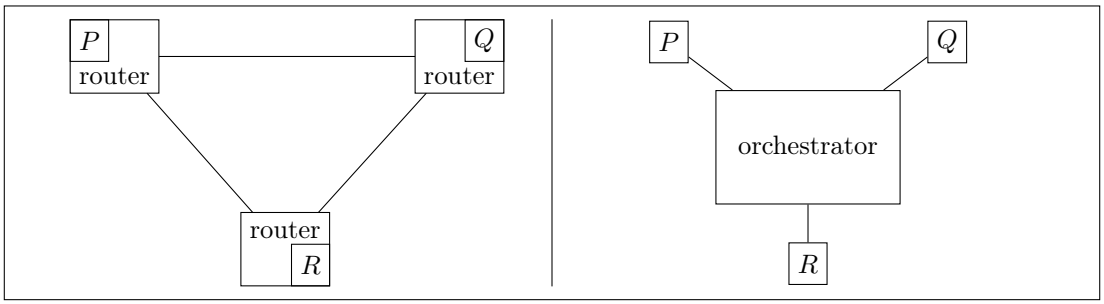

Figure 1: Given processes P, Q, and R implementing the roles of c, s, and a, respectively, protocol G_{auth} can be realized as a choreography (our approach, left) and as an orchestration (previous works, right).

Relative Projection and Dependencies To ensure that routed implementations conform to their global type, we again rely on the binary session type system. To verify the behavior on the channels that connect routed implementations, we use another form of projection: *relative projection*. Relative projection projects the global type onto *pairs of participants*, resulting in a view of the protocol that is *relative* only to these two participants. We use this relative view of the protocol as a binary session type to type check the communications on the channel connecting the routed implementations of this pair of participants.

A challenge in relative projection is the possibility of encountering *non-local choices*: choices made by other participants that affect the protocol between the two involved participants. We handle this by explicitely recording such non-local choices during relative projection as *dependencies*, informing the involved participants that they need to coordinate on the result of these non-local choices.

Related Work Its natural support for decentralized process networks is the distinguishing feature of our work with respect to similar prior works based on binary session type systems. Caires and Pérez [4] use centralized process networks: they connect each participant implementation to an additional orchestrator process derived from the global type. Carbone *et al.* [5] define a type system to validate choreographies of participant implementations using global types, but their deadlock freedom result relies on an encoding in which they also add a central orchestrator process. Compare the realizations of G_{auth} in Figure 1 as a decentralized (our approach, left) and as a centralized (prior works, right) process network.

Both these works crucially rely on orchestration, because they leverage on binary session type systems that only admit tree-shaped process networks. Hence, these type systems do not support the cyclic process networks needed for decentralization. Our work overcomes this obstacle by relying on a type system that does admit cyclic process networks: APCP [20]. APCP ensures deadlock freedom for well-typed processes by relying on annotations on types that prevent *circular dependencies*, such that processes never get stuck waiting for each other.

Other approaches to analyzing coordinations of interacting processes include using *Petri nets*. For example, Fossati *et al.* [10] express multiparty session protocols as *free choice* Petri nets [9], relying on classical results to prove protocol conformance and progress results for process implementations of such protocols.

Acknowledgments We thank the reviewers for their useful feedback. This work is partially supported by the Netherlands Organization for Scientific Research (NWO) under the VIDI Project No. 016.Vidi.189.046 (Unifying Correctness for Communicating Software).

References

[1] Franco Barbanera and Mariangiola Dezani-Ciancaglini. Open Multiparty Sessions. *Electronic Proceedings in Theoretical Computer Science*, 304:77–96, September 2019. arXiv:1909.05972, doi:10.4204/EPTCS.304.6.

[2] Andi Bejleri, Elton Domnori, Malte Viering, Patrick Eugster, and Mira Mezini. Comprehensive Multiparty Session Types. *The Art, Science, and Engineering of Programming*, 3(3):6:1–6:59, February 2019. doi:10.22152/programming-journal.org/2019/3/6.

[3] 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, *CONCUR 2008 - Concurrency Theory*, Lecture Notes in Computer Science, pages 418–433, Berlin, Heidelberg, 2008. Springer. doi:10.1007/978-3-540-85361-9_33.

[4] Luís Caires and Jorge A. Pérez. Multiparty Session Types Within a Canonical Binary Theory, and Beyond. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Distributed Objects, Components, and Systems*, Lecture Notes in Computer Science, pages 74–95. Springer International Publishing, 2016. doi:10.1007/978-3-319-39570-8_6.

[5] Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence Generalises Duality: A Logical Explanation of Multiparty Session Types. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory (CONCUR 2016)*, volume 59 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:15, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2016.33.

[6] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On Global Types and Multi-Party Session. *Logical Methods in Computer Science*, 8(1), March 2012. doi:10.2168/LMCS-8(1:24)2012.

[7] Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Ross Horne. Global types with internal delegation. *Theoretical Computer Science*, 807:128–153, February 2020. doi:10.1016/j.tcs.2019.09.027.

[8] Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *Mathematical Structures in Computer Science*, 26(2):238–302, February 2016. doi:10.1017/S0960129514000188.

[9] Jorg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge University Press, January 1995.

[10] Luca Fossati, Raymond Hu, and Nobuko Yoshida. Multiparty Session Nets. In Matteo Maffei and Emilio Tuosto, editors, *Trustworthy Global Computing*, Lecture Notes in Computer Science, pages 112–127, Berlin, Heidelberg, 2014. Springer. doi:10.1007/978-3-662-45917-1_8.

[11] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, pages 273–284, San Francisco, California, USA, January 2008. Association for Computing Machinery. doi:10.1145/1328438.1328472.

[12] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *Journal of the ACM*, 63(1), March 2016. doi:10.1145/2827695.

[13] Raymond Hu, Andi Bejleri, Nobuko Yoshida, and Pierre-Malo Denielou. Parameterised Multiparty Session Types. *Logical Methods in Computer Science*, Volume 8, Issue 4, October 2012. doi:10.2168/LMCS-8(4:6)2012.

[14] Keigo Imai, Rumyana Neykova, Nobuko Yoshida, and Shoji Yuen. Multiparty Session Programming With Global Protocol Combinators. In Robert Hirschfeld and Tobias Pape, editors, *34th European Conference on Object-Oriented Programming (ECOOP 2020)*, volume 166 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:30, Dagstuhl, Germany, 2020. Schloss

Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECOOP.2020.9.

[15] Rupak Majumdar, Nobuko Yoshida, and Damien Zufferey. Multiparty motion coordination: From choreographies to robotics programs. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA):134:1–134:30, November 2020. doi:10.1145/3428202.

[16] Luca Padovani, Vasco Thudichum Vasconcelos, and Hugo Torres Vieira. Typing Liveness in Multiparty Communicating Systems. In Eva Kühn and Rosario Pugliese, editors, *Coordination Models and Languages*, Lecture Notes in Computer Science, pages 147–162, Berlin, Heidelberg, 2014. Springer. doi:10.1007/978-3-662-43376-8_10.

[17] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, October 2003. doi:10.1109/MC.2003.1236471.

[18] Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming. In Peter Müller, editor, *31st European Conference on Object-Oriented Programming (ECOOP 2017)*, volume 74 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:31, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ECOOP.2017.24.

[19] Alceste Scalas and Nobuko Yoshida. Less is more: Multiparty session types revisited. *Proceedings of the ACM on Programming Languages*, 3(POPL):30:1–30:29, January 2019. Revised, extended version at https://www.doc.ic.ac.uk/research/technicalreports/2018/DTRS18-6.pdf. doi:10.1145/3290343.

[20] Bas van den Heuvel and Jorge A. Pérez. Deadlock Freedom for Asynchronous and Cyclic Process Networks. *Electronic Proceedings in Theoretical Computer Science*, 347:38–56, October 2021. arXiv:2110.00146, doi:10.4204/EPTCS.347.3.

[21] Bas van den Heuvel and Jorge A. Pérez. A Decentralized Analysis of Multiparty Protocols. *arXiv:2101.09038 [cs]*, June 2021. arXiv:2101.09038.

[22] W. M. P. van der Aalst. Orchestration. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 2004–2005. Springer US, Boston, MA, 2009. doi:10.1007/978-0-387-39940-9_1197.