# Choreographic Programming in Coq

Luís Cruz-Filipe, Lovro Lugović, Fabrizio Montesi, Marco Peressotti, and Robert R. Rasmussen *

Department of Mathematics and Computer Science, University of Southern Denmark
{lcf,lugovic,fmontesi,peressotti,rrr}@imada.sdu.dk

Choreographic programming is a paradigm for specifying concurrent systems (choreographies) based on message-passing where communications are written in an Alice-to-Bob notation. Choreographies can be mechanically *projected* into distributed process-calculus implementations guaranteed to be bisimilar to the original choreography. Such implementations can never suffer from mismatched communications – more generally, they cannot reach a deadlocked state, as the syntax of choreography language cannot express deadlocks.

**Example 1** (From [8])**.** *The following choreography models a scenario where Alice (*$\mathsf{a}$*) buys a book from a seller (*$\mathsf{s}$*) routing the payment through her bank (*$\mathsf{b}$*).*

$$\begin{aligned}&\mathsf{a}.\mathit{title} \to \mathsf{s};\ \mathsf{s}.\mathit{price} \to \mathsf{a};\ \mathsf{s}.\mathit{price} \to \mathsf{b};\\ &\mathit{if}\ \mathsf{b}.\mathit{approves}\ \mathit{then}\ \mathsf{b} \to \mathsf{s}[\mathit{ok}];\ \mathsf{b} \to \mathsf{a}[\mathit{ok}];\ \mathsf{s}.\mathit{book} \to \mathsf{a}\\ &\qquad \mathit{else}\ \mathsf{b} \to \mathsf{s}[\mathit{ko}];\ \mathsf{b} \to \mathsf{a}[\mathit{ko}];\ \mathbf{0}\end{aligned}$$

*First, Alice sends the title of the book to the seller, who quotes the price to both Alice and the bank. The bank can then confirm the transaction by sending an acknowledgement to both Alice and the seller (after which the latter sends the book), or send a cancellation to both parties.*

*This choreography can be projected into the following distributed protocol.*

$$\begin{aligned}&\mathsf{a} \triangleright \mathsf{s}!\mathit{title};\ \mathsf{s}?;\ \mathsf{b}\&\{\mathit{ok} :\ \mathsf{s}?, \mathit{ko} :\ \mathbf{0}\}\\ &\mathsf{b} \triangleright \mathsf{s}?;\ \mathit{if}\ \mathit{approves}\ \mathit{then}\ (\mathsf{s} \oplus \mathit{ok};\ \mathsf{a} \oplus \mathit{ok})\ \mathit{else}\ (\mathsf{s} \oplus \mathit{ko};\ \mathsf{a} \oplus \mathit{ko})\\ &\mathsf{s} \triangleright \mathsf{a}?;\ \mathsf{a}!\mathit{price};\ \mathsf{b}!\mathit{price};\ \mathsf{b}\&\{\mathit{ok} :\ \mathsf{a}!\mathit{book}, \mathit{ko} : \mathbf{0}\}\end{aligned}$$

*Alice's protocol is: send a title to the seller and wait for a reply; then wait for either confirmation from the bank, in which case the seller sends the book, or cancellation, in which case the protocol ends. The protocol for the seller is similar. In turn, the bank initially waits for a message from the seller, and then decides whether to send confirmation or cancellation to the seller and Alice.*

The precise operational correspondence between choreographies and their projections (the *EPP Theorem*) ensures that the choreography and the distributed protocol in the example above behave in the same way. The proof of this result is complex, due to the high number of cases that need to be considered and to the multitude of rules in the semantics of both choreography and process languages. Such proofs are prone to errors when designed and checked by humans: a previous attempt to formalise a higher-order process calculus [15] turned up a number of problems in the original proofs [16]; similar issues have arisen in the field of multiparty session types [20, Section 8.1], closely related to choreographic programming.

These issues motivated a subset of the present authors to formalise the theory of choreographic programming in the theorem prover Coq [8]. The result was a formalisation of a core model for choreographic programming [6, 17], including the choreographic language and proof

*This work was partially supported by Villum Fonden, grants 29518 and 50079, and Independent Research Fund Denmark, grant 0135-00219.

of its Turing completeness [10] and the target language for distributed implementations [6] together with a proof of the EPP Theorem [9]. In those works, we stated that our formalisation was developed with an intent to be extendable and flexible. In this abstract, we report on recent developments that build upon this formalisation, using it as an effective research tool, thereby establishing its reusability and its usefulness.

**Choreography amendment.** Not all choreographies can be projected to distributed implementations, because of a realisability requirement known in the field as *knowledge of choice* [3]. Essentially, this condition means that every process whose behaviour depends on a conditional expression evaluated by another process must be notified of this result. In the example above, this is achieved by the *label selection* communications, e.g., $\mathsf{b} \to \mathsf{s}[ok]$.

*Amendment* is a transformation that makes every choreography projectable by adding such label selections where needed. This procedure is described in [6]; however, the main correspondence result between the semantics of the original and the amended choreography is wrong. The error was only discovered while formalising the proof, and the counterexamples found with the theorem prover's help were essential to establishing and proving the correct correspondence [7].

**Livelocks.** Requiring knowledge of choice disallows choreographies where some participants are inactive while others engage in a loop – for example, if Alice and the seller engage in a negotiation until they agree on a price, after which the bank is notified of the amount to transfer. A direct formalisation of this protocol as a choreography would not be projectable: knowledge of choice would require the bank to be informed of the result of each iteration. This constraint is unreasonable in practice.

In recent work [11], we have relaxed this requirement to allow for projecting several new scenarios that occur in practice. The use of the theorem prover was again essential to detect edge cases that were not found while making pen-and-paper proofs. This development also supports the claim of modularity of the formalisation, as the proof of the EPP Theorem was mostly unchanged as soon as the relevant lemmas had been generalised to the new notion of projection.

**Compilation.** The distributed implementations generated by choreographic programming are written in a mathematical process language. However, they are close enough to implementation languages that they can be very directly translated to executable code.

We have implemented a toolchain that allows users to write choreographies, translates them in Coq terms, applies the projection procedure extracted from the Coq formalisation to obtain a distributed process implementation, and finally compiles this implementation into executable code [5]. The final compilation step is done by a handwritten program, but since it is completely homeomorphic (in the sense that each process action is modularly translated to Jolie code [18]) its correctness is easy to establish without requiring a full formalisation.

**Related work.** After our initial work, other groups have developed formalisations of choreographic and related languages. Kalas [19] is a certified compiler written in HOL from a choreographic language similar to ours to CakeML, with an asynchronous semantics but a more restricted notion of projection. In particular, processes evaluating conditionals must immediately send selections to the processes that need them, while our language is more faithful to the pen-and-paper literature on choreographies [1, 2, 13].

Pirouette [12] is a functional choreographic programming language formalised in Coq, supporting asynchronous communication and higher-order functions. These capabilities come at

the cost of hidden global synchronisations, while our language is fully decentralised, with all synchronisations syntactically explicit.

Another related line of research is that of multiparty session types [13], which can be seen as choreographies without computation – and therefore simpler. There are two available formalisations of multiparty session types [4, 14], which include a counterpart to the EPP theorem, but are even more restrictive than Kalas in how they project conditionals.

# References


[1] Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8:1–8:78, 2012.

[2] Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In Roberto Giacobazzi and Radhia Cousot, editors, *Procs. POPL*, pages 263–274. ACM, 2013.

[3] Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On global types and multi-party session. *Log. Methods Comput. Sci.*, 8(1), 2012.

[4] David Castro-Perez, Francisco Ferreira, Lorenzo Gheri, and Nobuko Yoshida. Zooid: a DSL for certified multiparty computation: from mechanised metatheory to certified multiparty processes. In Stephen N. Freund and Eran Yahav, editors, *Procs. PLDI*, pages 237–251. ACM, 2021.

[5] Luís Cruz-Filipe, Lovro Lugović, and Fabrizio Montesi. Certified compilation of choreographies with hacc. *CoRR*, abs/2303.03972, 2023. Accepted for publication at *FORTE'23*.

[6] Luís Cruz-Filipe and Fabrizio Montesi. A core model for choreographic programming. *Theor. Comput. Sci.*, 802:38–66, 2020.

[7] Luís Cruz-Filipe and Fabrizio Montesi. Now it compiles! certified automatic repair of uncompilable protocols. *CoRR*, abs/2302.14622, 2023. Accepted for publication at *ITP'23*.

[8] Luís Cruz-Filipe, Fabrizio Montesi, and Marco Peressotti. Choreographies in Coq. In *TYPES 2019, Abstracts*, 2019. Extended abstract.

[9] Luís Cruz-Filipe, Fabrizio Montesi, and Marco Peressotti. Certifying choreography compilation. In Antonio Cerone and Peter Csaba Ölveczky, editors, *Procs. ICTAC*, volume 12819 of *LNCS*, pages 115–133. Springer, 2021.

[10] Luís Cruz-Filipe, Fabrizio Montesi, and Marco Peressotti. Formalising a Turing-complete choreographic language in Coq. In Liron Cohen and Cezary Kaliszyk, editors, *Procs. ITP*, volume 193 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[11] Luís Cruz-Filipe, Fabrizio Montesi, and Robert R. Rasmussen. Keep me out of the loop: a more flexible choreographic projection. Accepted for publication., 2023.

[12] Andrew K. Hirsch and Deepak Garg. Pirouette: higher-order typed functional choreographies. *Proc. ACM Program. Lang.*, 6(POPL):1–27, 2022.

[13] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9, 2016. Also: POPL, pages 273–284, 2008.

[14] Jules Jacobs, Stephanie Balzer, and Robbert Krebbers. Multiparty gv: Functional multiparty session types with certified deadlock freedom. In *Procs. ICFP*, 2022. Accepted for publication.

[15] Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. *Inf. Comput.*, 209(2):198–226, 2011.

[16] Petar Maksimovic and Alan Schmitt. HOCore in Coq. In Christian Urban and Xingyuan Zhang, editors, *ITP*, volume 9236 of *LNCS*, pages 278–293. Springer, 2015.

[17] Fabrizio Montesi. *Introduction to Choreographies*. Cambridge University Press, 2023.

[18] Fabrizio Montesi, Claudio Guidi, and Gianluigi Zavattaro. Service-oriented programming with jolie. In Athman Bouguettaya, Quan Z. Sheng, and Florian Daniel, editors, *Web Services Foun-*

*dations*, pages 81–107. Springer, 2014.

[19] Johannes Åman Pohjola, Alejandro Gómez-Londoño, James Shaker, and Michael Norrish. Kalas: A verified, end-to-end compiler for a choreographic language. In June Andronick and Leonardo de Moura, editors, *Procs. ITP*, volume 237 of *LIPIcs*, pages 27:1–27:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[20] Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *PACMPL*, 3(POPL):30:1–30:29, 2019.