On choreographic programming and lossy communications

Marco Peressotti

Department of Mathematics and Computer Science and Physics, University of Southern Denmark
peressotti@imada.sdu.dk

Abstract

Choreographic Programming [12] is an emerging paradigm for programming communications in concurrent and distributed systems. The key idea is that programs are *choreographies*, which define the communications that we wish to take place from a global viewpoint, using structures inspired by the "Alice and Bob" notation for security protocols [14]. Then, an *EndPoint Projection* (EPP) synthesises a correct-by-construction implementation in process models (*e.g.*, process calculi), guaranteeing important properties such as progress and operational correspondence [1, 2]. The applicability of the paradigm has been demonstrated in different settings, including service-oriented programming [1, 2, 13, 3], adaptable distributed software [6], cyber-physical systems [10, 9], and software verification [4].

Processes in choreographic programs typically interact via *reliable* point-to-point message passing. This is expressed by language terms like $\mathsf{p}.e \rightarrow \mathsf{q}.y$, which reads "process p evaluates expression e locally and sends the result to process q, which stores the received value in its local variable y". However, this is not quite what happens in reality since a) communication is (usually) asynchronous and b) reliability is not absolute but approximated.

The first issue has been addressed in a series of works—we mention [7, 8, 5]. Intuitively, a communication $\mathsf{p}.e \rightarrow \mathsf{q}.y$ can be thought as happening in two steps one where p sending the message and one where the message is received by q *e.g.*, by the runtime terms $\mathsf{p}.e \rightarrow \underline{x}; \underline{x} \rightarrow \mathsf{q}.y$ where the fresh name x is replaced by the message in transit.

This is the first work to investigate the second issue in a choreographic setting[1]. We focus on communication channels where sending and receiving operations may fail—it is fair to assume that receiving can succeed only if so does sending. We ignore the issue of message duplication or modification during transit. We do not assume any particular model or mechanism for failure detection or propagation of this information. Reworded, to assume failures to be consistently known to all participants would be misleading. As a consequence, choreographies can only rely on knowledge local to each process when reacting and recovering from communication failures.

Observe that in the classical choreographic primitive $\mathsf{p}.e \rightarrow \mathsf{q}.y$ *communication intention* from *communication implementation* are fused and indistinguishable. In the setting considered in this work instead, both ends of a communication may need to react to failures and retry their action (send/receive) or engage in additional interactions (*e.g.*, with a logger). The local nature of failures and the need to program recovering strategies calls for new communication primitives that separate all these phases. To this end, we replace the classical primitive $\mathsf{p}.e \rightarrow \mathsf{q}.y$ with a primitive for declaring a communication $i : \mathsf{p} \rightarrow \mathsf{q}$, attempting its send action $\mathsf{p}.e \rightarrow i$ and its receive action $i \rightarrow \mathsf{q}.y$ where i is a communication identifier bound by the declaration and all other terms are as in $\mathsf{p}.e \rightarrow \mathsf{q}.y$. Observe that the sent value and the location where it is stored may change at each attempt *e.g.*, to count failures. We assume that each process can query the outcome of an attempt from a local variable associated to the communication identifier. As an example, consider the following fragment:

[1]Perhaps the closest work to ours is [11] where a choreographic model with reversible computations is proposed. We remark that in *loc. cit.* communication is assumed lossless in order to coordinate recovery phases and communication roll-backs.

```
1 i : p -> q;
2 def RECV :
3    i -> q.y;
4    if i@q then 0 else RECV in
5 def SEND :
6    p.e -> i;
7    if i@p then RECV else SEND in
8 SEND
```

At line 1 a communication from p to q is declared and denoted by i. At line 2 starts the recursive definition of $RECV$ where an attempt to receive is made (line 3) before either completing the protocol or retrying the operation depending on the status of i (Line 4). At line 5 starts the definition of $SEND$ which behaves likewise. Finally, at line 8, the choreography starts protocol $SEND$. Intuitively, this elementary program implements a reliable communication from p to q by repeatedly attempting each send and receive until the message is successfully delivered. In order to formally prove that his and similar programs do actually succeed in implementing their communications we introduce a probabilistic semantics of choreographic programs to support probabilistic reasoning. This allows us to formalise and answer questions like “how likely is a protocol to deadlock given a certain probability of communication failure”.

References

[1] M. Carbone, K. Honda, and N. Yoshida. Structured communication-centered programming for web services. *ACM Trans. Program. Lang. Syst.*, 34(2):8, 2012.

[2] M. Carbone and F. Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *POPL*, pages 263–274. ACM, 2013.

[3] Chor. Programming Language. `http://www.chor-lang.org/`.

[4] L. Cruz-Filipe, K. S. Larsen, and F. Montesi. The paths to choreography extraction. In J. Esparza and A. S. Murawski, editors, *FoSSaCS*, volume 10203 of *LNCS*, pages 424–440. Springer, 2017.

[5] L. Cruz-Filipe and F. Montesi. Encoding asynchrony in choreographies. In *SAC*, pages 1175–1177. ACM, 2017.

[6] M. Dalla Preda, M. Gabbrielli, S. Giallorenzo, I. Lanese, and J. Mauro. Dynamic choreographies: Theory and implementation. *Logical Methods in Computer Science*, 13(2), 2017.

[7] K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.

[8] R. Kazhamiakin and M. Pistore. Choreography conformance analysis: Asynchronous communications and information alignment. In *WS-FM*, pages 227–241, 2006.

[9] H. A. López and K. Heussen. Choreographing cyber-physical distributed control systems for the energy sector. In *SAC*, pages 437–443. ACM, 2017.

[10] H. A. López, F. Nielson, and H. R. Nielson. Enforcing availability in failure-aware communicating systems. In *FORTE*, volume 9688 of *Lecture Notes in Computer Science*, pages 195–211. Springer, 2016.

[11] C. A. Mezzina and E. Tuosto. Choreographies for automatic recovery. *CoRR*, abs/1705.09525, 2017.

[12] F. Montesi. *Choreographic Programming.* Ph.D. Thesis, IT University of Copenhagen, 2013.

[13] F. Montesi and N. Yoshida. Compositional choreographies. In *CONCUR*, volume 8052 of *LNCS*, pages 425–439. Springer, 2013.

[14] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, Dec. 1978.