Choreographies for Microservices

Saverio Giallorenzo and Ivan Lanese

Università di Bologna, Bologna, Italy / INRIA

Introduction. Microservices (MS) are cohesive [1] independent software entities that communicate via message passing [10]. Cohesiveness of microservices goes hand-in-hand with loose coupling, making the creation, deployment, and evolution of MS Architectures (MSA) flexible. In particular, MSAs are famous for optimal scalability (only the microservices actually interested by a surge of inbound requests are replicated) and continuous integration (fixing an MSA does not imply to redeploy the whole architecture but just a subset of its microservices).

Since basic microservices focus on exposing simple, cohesive functionalities, any complex behaviour offered by an MSA results from a composition of a large number of microservices. Hence, in MSAs interactions among microservices are usually more complex than in traditional distributed systems, which makes their coordination prone to errors like races or deadlocks [7], resulting in hard-to-debug misbehaviours and blocks. To guarantee that an MSA adheres to its expected behaviour, programmers need a single artefact that condenses the emerging behaviour of the composition. Usually, programmers have to create such a single, unifying picture over the MSA manually, following a time-consuming and error-prone activity that reconstructs the behaviour of the architecture from the single interactions among the constituent microservices.

Contribution. Our intuition is that choreographies — originally designed to specify distributed interactions [12] from a global point of view — can structure and automatise the creation and inspection of MSAs. Hence, in this presentation, we start a novel discussion on how different interpretations of choreographies can aid the development of correct MSAs, i.e., those that faithfully implement a desired distributed interaction, also avoiding deadlocks and races. To this aim, in the following, we provide a brief background on choreographies, discuss four of their possible usages in the context of microservices, and conclude with prospective work.

Background. Choreographies describe the communications among the components of a distributed system in a unique artefact; they are a blueprint of the desired behaviour of a distributed system. The main feature of a choreography is that communications are described in terms of interactions, coupling each output to its corresponding input. The syntax follows the form $m : A \to B$, which reads "message m is sent by component A and received by component B". In a choreography, interactions can be composed (e.g., in sequence, parallel, choice, within loops, etc.) to specify the order of the exchanges among the components in a distributed system.

Choreographies for Microservices. We now discuss the main approaches based on choreographies, and their possible usages in MSA development. Common denominator of all interpretations is that communication deadlocks and races are avoided by construction. First we consider two dimensions to categorise choreography usages and discuss them later on:

- *Top-down vs Bottom-up* considers choreographies as start- or end-point of a workflow: top-down choreographies can be used as blueprints to automatically generate the description of each participant; bottom-up choreographies work as support tool to synthesise (in an automatic way) the actual composition of all the participants, given their description.

- *Choreographies-as-programs vs Choreographies-as-types* refers to the kind of information contained in the choreography: choreographies-as-programs contain the full description of the system; choreographies-as-types abstract the system, focusing on the patterns of interactions expressed as behavioural types [4].

Top-down, Choreographies-as-programs. In the top-down usage [2, 11], an MSA is built by defining its behaviour in terms of a choreography. Then, the choreography is compiled into a set of microservices, possibly including the automatic generation of deployment information, e.g., their bindings or instructions to include them in their respective containers [8]. In case of updates, to preserve the correctness properties of choreographies, programmers need to modify the original choreography and redeploy the recompiled microservices. Top-down choreographies provide a clear description of the MSA, which follows the specified behaviour by construction. The main issue here is that the choreography needs to include all the details (and complexity) of the MSA. This approach is optimal to create new systems of microservices but not to modify existing ones, since it requires the mapping of all existing microservices into the choreography.

Bottom-up, Choreographies-as-programs. Ideally, bottom-up choreographies are synthesised from the code of existing microservice systems. In case the MSA has communication errors, the choreography cannot be generated, essentially detecting communication errors and providing useful information to correct the issue. Bottom-up choreographies can in turn be used in other approaches, e.g., they can be modified and used as top-down choreographies to generate the updated code for microservices. At the moment, there are no implementations of bottom-up choreographies-as-programs that can deal with actual code. Indeed, the bottom-up approach is currently supported only for abstract descriptions of microservice systems.

Top-down, Choreographies-as-types. In this case, the choreography contains an abstract description of the system, focusing on the communication behaviour [3]. Then, the top-down choreography can be used to generate the specification of the expected communication behaviour of each microservice. Once generated, the local specifications can be used *i*) to verify the correctness of the local programs, either statically (type checking) or dynamically (monitoring), or *ii*) as a skeleton to actually write the code of the microservice.

Bottom-up, Choreographies-as-types. Choreographies-as-types can also be synthesised from the behavioural types of the local programs [5], or from analogous descriptions such as communicating automata [6], which may be available in some development processes, or can be extracted from the code of microservices. Synthesis can be used to check for compliance among MSs.

Conclusion and Future Work. Choreographies can become an essential tool for MS development, however some enhancements are still needed to bring them to a wider audience. A main issue is that choreographies mainly describe static, closed systems, however approaches to both adaptive [11] and open choreographies [9] are now appearing, promising a higher degree of flexibility in MS development. Another issue is that the relation between choreographies and MSA deployment has not been explored in any depth, yet developers of MSAs could sensibly benefit from top-down approaches that automatise the containerisation phase of MS deployment.

References

[1] James M. Bieman and Byung-Kyoo Kang. "Cohesion and Reuse in an Object-oriented System". In: *SSR*. ACM, 1995, pp. 259–262. ISBN: 0-89791-739-1.

[2] Marco Carbone and Fabrizio Montesi. "Deadlock-freedom-by-design: multiparty asynchronous global programming". In: *POPL*. ACM, 2013, pp. 263–274.

[3] Kohei Honda et al. "Multiparty Asynchronous Session Types". In: *J. ACM* 63.1 (Mar. 2016), 9:1–9:67. ISSN: 0004-5411.

[4] Hans Hüttel et al. "Foundations of session types and behavioural contracts". In: *ACM Computing Surveys* 49.1 (2016), p. 3.

[5] Julien Lange and Emilio Tuosto. "Synthesising Choreographies from Local Session Types". In: *CONCUR*. Springer, 2012, pp. 225–239.

[6] Julien Lange et al. "From Communicating Machines to Graphical Choreographies". In: *POPL*. ACM, 2015, pp. 221–232.

[7] Shan Lu et al. "Learning from mistakes: a comprehensive study on real world concurrency bug characteristics". In: *ACM Sigplan Notices*. Vol. 43. 3. ACM. 2008, pp. 329–339.

[8] Dirk Merkel. "Docker: lightweight linux containers for consistent development and deployment". In: *Linux Journal* 2014.239 (2014), p. 2.

[9] Fabrizio Montesi and Nobuko Yoshida. "Compositional Choreographies". In: *CONCUR*. Vol. 8052. LNCS. Springer, 2013, pp. 425–439.

[10] Sam Newman. *Building Microservices*. "O'Reilly Media, Inc.", 2015.

[11] Mila Dalla Preda et al. "Dynamic Choreographies: Theory And Implementation". In: *Logical Methods in Computer Science* 13.2 (2017).

[12] *Web Services Choreography Description Language Version 1.0*. http://www.w3.org/TR/ws-cdl-10/. W3C. 2005.