Ballerina and Jolie: Connecting Two Frontiers of Microservice Programming

Anjana Fernando[1], Saverio Giallorenzo[2], Claudio Guidi[3], Sameera Jayasoma[1], Balint Maschio[4], Jacopo Mauro[2], Fabrizio Montesi[2], Marco Montesi[5], Marco Peressotti[2], Matthias Dieter Wallnöfer[6], and Lakmal Warusawithana[1]

[1] WSO2
[2] University of Southern Denmark
[3] italianaSoftware s.r.l.
[4] Pixis co.
[5] Teamsystem SpA
[6] Technologische Fachoberschule "Max Valier" Bozen

1 Introduction

Cloud Computing and containerization are changing the way we conceive software artefacts. In the last years, they had a big impact at the infrastructure level, by facilitating the offer of virtual computational resources both in the form of virtual machines and containerization. In particular, containerization technologies offer an abstraction of computational resources that is no more related to the idea of a machine.

Containers facilitate the adoption of different technologies in a cloud environment, which has motivated the application of different tools and languages for each specific microservice. Nevertheless, in the programming of microservices, some common aspects always emerge, independently of the specific microservice that is being developed. These aspects include, for example, the programming of communications, monitoring, fault management, and architectural patterns (like API gateway).

Over the years, the teams behind the programming languages Ballerina [2] and Jolie [3] have developed linguistic primitives to deal with these aspects. The starting point of our presentation is: while the two languages have been developed independently, many of their features are strikingly similar. This realisation came out of a recent meeting between the two teams, and inspires the question:

Is it a coincidence, or are we facing a new generation of programming languages?

We outline what we will discuss in our presentation.

- The common programming concepts for microservice programming that we have identified in Ballerina and Jolie, and how they connect to previous work. For example, these concepts are strongly linked to established notions in computer science (like process calculi and types) [10, 16, 17, 9] and computer engineering (like programming in the large versus programming in the small and workflows) [5, 15, 18].
- How these programming concepts are supported linguistically by Ballerina and Jolie.
- Engage with the audience on our previous question.

We give an overview of the identified programming aspects in the next section.

2 Microservice programming concepts

We list the concepts of microservice programming that we will present. These concepts support the overarching concern that microservice languages should aid in the analysis and understanding of complex systems.

Service orientation Services are native constructs, and services can be compositions of other services [4, 13].

Communication primitives Communication actions (like send and receive) are supported by native primitives [4, 12].

Manifest workflows The workflows enacted by a service are made explicitly manifest by the language [6, 11].

Access points The access points by which a service can receive messages are declared explicitly [4, 13].

Dependencies The dependencies of a service on other services are declared explicitly [13].

APIs Access points and dependencies are typed with interfaces that include the types of messages that can be exchanged [4, 13].

Message types and values Types and values are designed with network transmission in mind (marshalling/unmarshalling) [8, 13].

Interoperability Interoperability with other technologies over network communications and different transport protocols is a first-class citizen [1, 11].

Architectural design extraction The overall architecture of a complex microservice system should be automatically extractable.

Architectural programming Common architectural design patterns like proxy-based access control are natively supported [14].

Built-in observability A service always offers a way to monitor their states and the actions that they perform, also remotely [7].

References

[1] Ballerina HTTP module. https://ballerina.io/learn/api-docs/ballerina/http/index.html, 2020.

[2] Ballerina website. https://ballerina.io/, 2020.

[3] Jolie website. https://jolie-lang.org/, 2020.

[4] J. Clark, S. Weerawarana, and H. Aravinda. Ballerina Language Specification, 2020R1. https://ballerina.io/spec/lang/2020R1/, 2020.

[5] F. DeRemer and H. H. Kron. Programming-in-the-large versus programming-in-the-small. *IEEE Trans. Software Eng.*, 2(2):80–86, 1976.

[6] A. Fernando. Making Sequence Diagrams Cool Again. https://hackernoon.com/rethinking-programming-making-sequence-diagrams-cool-again-6z1p3yv9, 2020.

[7] A. Fernando. Rethinking Programming: Automated Observability. https://hackernoon.com/rethinking-programming-automated-observability-dn14p3yxb, 2020.

[8] A. Fernando. Rethinking Programming: Network-Aware Type System. https://hackernoon.com/rethinking-programming-network-aware-type-system-8o7x3yh6, 2020.

[9] F. Leymann and D. Roller. Modeling business processes with BPEL4WS. *Inf. Syst. E-Business Management*, 4(3):265–284, 2006.

[10] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[11] F. Montesi. Process-aware web programming with Jolie. *Sci. Comput. Program.*, 130:69–96, 2016.

[12] F. Montesi, C. Guidi, R. Lucchi, and G. Zavattaro. JOLIE: a java orchestration language interpreter engine. *Electron. Notes Theor. Comput. Sci.*, 181:19–33, 2007.

[13] F. Montesi, C. Guidi, and G. Zavattaro. Service-Oriented Programming with Jolie. In A. Bouguettaya, Q. Z. Sheng, and F. Daniel, editors, *Web Services Foundations*, pages 81–107. Springer, 2014.

[14] F. Montesi and J. Weber. From the decorator pattern to circuit breakers in microservices. In H. M. Haddad, R. L. Wainwright, and R. Chbeir, editors, *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*, pages 1733–1735. ACM, 2018.

[15] C. Pautasso and G. Alonso. Jopera: A toolkit for efficient visual composition of web services. *Int. J. Electron. Commer.*, 9(2):107–141, 2005.

[16] B. C. Pierce and C. Benjamin. *Types and programming languages*. MIT press, 2002.

[17] W. M. P. van der Aalst. Verification of workflow nets. In P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997, 18th International Conference, ICATPN '97, Toulouse, France, June 23-27, 1997, Proceedings*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer, 1997.

[18] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed Parallel Databases*, 14(1):5–51, 2003.