Microservice-Oriented Computing for the Internet of Things

Maurizio Gabbrielli, Ivan Lanese, and Stefano Pio Zingaro

Università di Bologna / INRIA Sophia Antipolis

1 Introduction and Contribution

Microservices [1] are a new trend in software architecture, advocating for building modern large scale software systems using a flexible composition of small, independent services.

Internet of Things (IoT) [2] promotes the use of multi-layered platforms including low-level Things, such as sensors and actuators, edge devices, such as Fog nodes, and Cloud services. Because of IoT current hype, multiple works tackled the related issues of interactions in a heterogeneous and dynamically changing environment. Some recent approaches leverage the usage of service-oriented solutions in IoT scenarios [3]. These appraches are particularly well-suited since both service-oriented systems and IoT involve integration of heterogeneous systems. For the same reasons, we believe that fine-grained microservice-oriented approaches can further improve the added value of IoT applications.

In particular, IoT applications development becomes harder when integration is needed not just between different platforms — each adopting its own media protocols and data formats — but, also between different layers, e.g. Things with Fog, Fog with Cloud, etc. The integration involves many levels of the communication stack, spanning from link-layer communication technologies, such as ZigBee and WiFi, to application-layer protocols like HTTP, CoAP, and MQTT, reaching the top-most layers of data-format representation [2].

In this work we present the advantages of adopting a microservice-oriented solution to tackle the problem of IoT integration (both cross-layer and cross-platform). We will follow a language-based approach focusing on integration at both the transport and application layer.

2 A Microservice-Oriented Language for IoT

We could tackle the problems mentioned above from different directions, e.g., by creating a library for a general-purpose programming language, by developing a new language or framework, or by building on top of an existing solution. We choose the latter approach, to leverage work done in service- and microservice-oriented computing in the last years. Many tools aim at supporting developers in the definition of technology-agnostic services and we selected one of them — the Jolie programming language [4] — because it *i*) let different communication protocols seamlessly coexist and interoperate within the same program and *ii*) let programmers dynamically choose which communication stack should be used for any given communication.

Since Jolie comes from the area of Service-Oriented Architectures [5], it already supports many web-oriented protocols (e.g., TCP at the transport level, SOAP, RMI and HTTP at the application level). These protocols are also the preferred communication technologies for Cloud and Fog computing scenarios. However, IoT applications rely not only on the technologies above, but they also need to support the most used communication stacks at the *Things* layer. Those are based on lightweight and connection-less protocols such as CoAP over UDP, respectively at application and tranport level, or asynchronous publish/subscribe protocols such as

MQTT. Thus, we focused on their integration in the Jolie interpreter. Notably, when the application protocol supports different data formats (such as JSON, XML, etc.) for the message payload, as in the case of HTTP and CoAP, the proposed solution is able to automatically marshal and un-marshal data as required. Concretely, we forked the Jolie interpreter into a prototype called JIoT, standing for "Jolie for IoT" including support for CoAP over UDP and MQTT. For further details on JIoT we refer to [6], from which this work takes large part of its contribution.

We now give an overview on how protocols are implemented in Jolie, to clarify the steps we had to take to add the protocols above. In Jolie the implementations of the supported application and transport protocols are independent. This enables the composition of any transport protocol with any application protocol. Being written in Java, the Jolie interpreter provides proper abstract classes that represent application and transport protocols. Each protocol is obtained as an implementation of the corresponding abstract classes. Each implementation is a separated library which is loaded only if the protocol is used. The main challenge has been integrating MQTT publish/subscribe into the Jolie framework, which is designed to support end-to-end communications.

3 Illustrative Example

In Listing 1, we illustrate the proposed approach considering a common "edge computing" scenario in IoT applications development, where we need to collect temperature data from three islands of IoT devices, relying on different communication stacks, for further elaboration.

Listing 1 highlights how, using the proposed language abstractions, the programmer can write a unique behaviour and exploit it to receive data sent over heterogeneous technology stacks. The microservice is composed of two parts: a *behaviour* (lines 25-29) specifying the logic of the elaboration (here we simply show the receive of temperature data), and a *deployment*, describing in a declarative way how communication is performed (here, either on `http` at lines 5-9, `coap` at lines 11-15, or `mqtt` at lines 17-23). This separation of concerns is fundamental to let programmers easily change which communication stack to use, preserving the same logic for the elaboration.

```
1  interface TemperatureInterface {
2    OneWay: receiveTemperature( string )
3  }
4
5  inputPort CollectorPort1 {
6    Location: "socket://localhost:8080"
7    Protocol: http
8    Interfaces: TemperatureInterface
9  }
10
11 inputPort CollectorPort2 {
12   Location: "datagram://localhost:5683"
13   Protocol: coap
14   Interfaces: TemperatureInterface
15 }
16
17 inputPort CollectorPort3 {
18   Location: "socket://localhost:9001"
```

```
19   Protocol: mqtt {
20     .broker = "socket://localhost:1883"
21   }
22   Interfaces: TemperatureInterface
23 }
24
25 main {
26   //...
27   receiveTemperature( data );
28   //...
29 }
```

Listing 1: Snippets of code with deployments (lines 1–23) and behaviour (lines 25–29) data.

4 Final Considerations

Several ongoing European projects, e.g. Inter-IoT [7], tackle the problem of collaboration and integration among heterogeneous systems. Technology-wise, architects of these systems can choose between two approaches: either favouring optimal in-layer communications by allowing each layer to select a different communication technology, or favouring cross-layer consistency. The former approach is weak for cross-layer integration, due to heterogeneity and possible incompatibility of the chosen standards. The latter makes the integration simpler thanks to the adoption of a single medium and data format. However, such enforced uniformity causes the application to become a *silo* providing little support for cross-platform integration. In this work we proposed a solution tackling both the mentioned problems by adopting a microservice-oriented computing approach to overcome cross-platform and inter-layer integration issues.

References

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: Yesterday, today, and tomorrow," *Present and Ulterior Software Engineering*, pp. 195–216, 2017.

[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys and Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[3] B. Butzin, F. Golatowski, and D. Timmermann, "Microservices approach for the Internet of Things," *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, vol. 2016-Novem, 2016.

[4] F. Montesi, C. Guidi, and G. Zavattaro, "Composing services with Jolie," *Proceedings of the 5th IEEE European Conference on Web Services, ECOWS 07*, pp. 13–22, 2007.

[5] T. Erl, *Soa: principles of service design*, vol. 1. Prentice Hall Upper Saddle River, 2008.

[6] M. Gabbrielli, S. Giallorenzo, I. Lanese, and S. Zingaro, "A language-based approach for interoperability of IoT platforms," in *HICSS*, AieSel, 2018.

[7] M. Ganzha, M. Paprzycki, W. Pawlowski, P. Szmeja, and K. Wasielewska, "Semantic technologies for the IoT - an Inter-IoT perspective," *Proceedings - 2016 IEEE 1st International Conference on Internet-of-Things Design and Implementation, IoTDI 2016*, pp. 271–276, 2016.