Institute of Architecture of Application Systems

Unified Integration of Smart Contracts Through Service Orientation

Ghareeb Falazi[1], Andrea Lamparelli[2], Uwe Breitenbücher[1], Florian Daniel[2], Frank Leymann[1]

[1]Institute of Architecture of Application Systems, University of Stuttgart, Germany
{falazi, breitenbuecher, leymann}@iaas.uni-stuttgart.de

[2]Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Italy
andrea.lamparelli@mail.polimi.it, florian.daniel@polimi.it

BibTeX:

```
@article {Falazi2020_UnifiedIntegrationBlockchains,
    Author    = {Ghareeb Falazi and Andrea Lamparelli and Uwe Breitenb{\"u}cher
                 and Florian Daniel and Frank Leymann},
    Title     = {{Unified Integration of Smart Contracts Through Service
                  Orientation}},
    Journal   = {IEEE Software},
    Publisher = {IEEE},
    Volume    = {37},
    Number    = {5},
    Year      = 2020,
    doi       = {10.1109/MS.2020.2994040}
}
```


This is the accepted version of the article, the final published version can be accessed at: https://doi.org/10.1109/MS.2020.2994040

Universität Stuttgart
Germany

Department: Blockchain and Smart Contract Engineering
Editor: Name, xxxx@email

Unified Integration of Smart Contracts through Service Orientation

G. Falazi
University of Stuttgart

A. Lamparelli
Politecnico di Milano

U. Breitenbücher
University of Stuttgart

F. Daniel
Politecnico di Milano

F. Leymann
University of Stuttgart

***Abstract*—This article introduces the reader to a set of technologies that lay the foundation for a service-oriented integration of smart contracts into generic software applications, such as business processes or enterprise applications. Using a typical supply chain scenario, the article showcases the use of the Smart Contract Description Language (SCDL) to describe the external interfaces of smart contracts, the Smart Contract Locator (SCL) to locate contracts deployed inside blockchain networks, and the Smart Contract Invocation Protocol (SCIP) to interact with them from the outside of the blockchain networks. The three specifications abstract away from blockchain specifics, provide developers with a unified view over multiple, heterogeneous blockchain technologies, and are supported by a reference implementation of a SCIP endpoint able to automatically turn abstract interactions into blockchain-specific ones.**

SINCE THE INTRODUCTION of flexible smart contracts – commonly associated with the birth of Ethereum (https://ethereum.org) – blockchains have expanded their applicability way beyond cryptocurrencies into application scenarios as diverse as supply chain management, healthcare, IoT, data management, and similar [1]. Smart contract-enabled blockchains today are full-fledged, distributed computing platforms [2] that are able to run application code inside the blockchain and to equip applications running out-

side the blockchain with trustworthy, deterministic on-chain functionalities, such as payment or logging services.

Integrating smart contracts into software that runs outside the blockchain is however not straightforward: each blockchain comes with proprietary transaction logic, consensus mechanisms, APIs, communication protocols, data formats, and authentication models. To invoke a given smart contract, one must master all these aspects and have access to a node of the blockchain network. And this must be repeated for each blockchain one wants to work with. If an application involves multiple blockchains, complexity thus grows significantly, if not prohibitively.

Orthogonally, we observe a general lack of software engineering technologies, tools and methodologies that take into account the conceptual specifics of blockchains and smart contracts (e.g., the lack of a guarantee that a transaction is durably written on the blockchain) and, at the same time, abstract away from technological details to help developers reuse smart contracts.

To address these shortcomings, in prior work, we conceived three ingredients to interpret smart contracts as building blocks of a service-oriented architecture: a smart contract description language and a smart contract locator [3], as well as an abstract interaction protocol [4]. In this article, we provide a holistic view on these technologies and, with the help of a supply chain scenario, showcase how their joint use eases the integration of smart contracts into generic software applications.

BACKGROUND

Service-Orientation for Smart Contracts

Service orientation is the philosophy underlying service-oriented computing [5]. It promotes a software ecosystem, the Service-Oriented Architecture (SOA), for the reuse of application logic – *web services* or *web APIs* – among nodes of a network, typically the Internet. The SOA is based on three roles: *providers* provision services, *registries* advertise services, and *consumers* discover and invoke services. Services are addressed using standard *URLs* and invoked using either XML-based SOAP or plain HTTP *messages*.

Smart contracts are different. They run inside blockchain networks, do not use the standard stack of web protocols, and cannot be addressed from the Internet. However, they have functions and can emit events like web services [2], somebody provisions them, and somebody uses them. If we use a gateway to mediate between blockchain networks and the Internet, it is possible to access smart contracts like services.

In our prior work, we laid the foundation for such Internet-accessible smart contracts and defined specifications to abstract away from technologies. Providers can describe the external interface of contracts (functions, events, parameters) using the JSON-based Smart Contract Description Language (SCDL) [3]. Contracts can be addressed by consumers using a Smart Contract Locator (SCL) [3], which is composed of a standard URL addressing the gateway and a blockchain-specific address of the contract. And contracts can be invoked using the Smart Contract Invocation Protocol (SCIP) [4], a JSON-RPC based message format to invoke a function, subscribe to events, receive callbacks, and query for events. See Figure 1 for an example of an SCL address and a SCIP message.

Related Works

The problem of connecting or accessing smart contracts of different blockchains is not new and has been approached differently so far.

Blockchain interoperability, as proposed by Interledger (https://interledger.org), Polkadot (https://polkadot.network) or Lightning Network (https://lightning.network/), focuses on enabling blockchains to interact with each other. The idea of *blockchain gateways* was introduced by Thomas et al. [6], however still in the context of blockchain interoperability and not for generic applications. For this purpose, *connector-based approaches* like Unibright [7] propose full-fledged platforms to communicate with blockchains, on the one hand, and with blockchain-external applications via extensible connectors, on the other hand. The use of vendor-specific platforms and connectors, however, risks to turn the technical integration problem into a product selection or marketing problem. The *Web Ledger Protocol* 1.0 [8] proposes a generic data model and syntax for blockchains and a so-called Ledger Agent HTTP API to create transactions,

append data, and query the blockchain; the protocol, however, does not propose dedicated abstractions for interacting with smart contracts.

Finally, Xu et al. [9] consider blockchains as *software connectors* that can provide external applications with communication, coordination, conversion and facilitation services. Smart contracts are used as middleware to connect applications, not as reusable services.

All these works raise the need for communicating with smart contracts, but only the three specifications we showcase in this article aim to provide an open specification for uniform smart contract integration that everyone can implement for free.

SMART CONTRACT INTEGRATION

In order to showcase the joint use of SCL, SCDL and SCIP, we discuss their application in the context of food supply chains.

In recent years, food product safety scandals have caused a decrease in customer confidence towards the quality and originality of food products [10]. Much of the problem can be attributed to the complexity of the structure of food supply chains, which results in the lack of transparency and provenance knowledge [11]. Blockchain technology offers a viable distributed platform that allows to enhance the knowledge of product provenance by customers and partners alike. This is achieved by its unique ability to irreversibly and immutably store the history of transactions, which guarantees supply chain participants the ability to trace origin, certify authenticity, track custody and verify integrity of products [12].

Food Supply Chains Scenario

Figure 1 shows two simplified supply chains for seafood and dairy products. The seafood supply chain starts with individual fishermen, or commercial fisheries, which catch seafood resources like fish. Next, fish are transported via shipping firms to their first buyers overseas, which process and package them appropriately. Finally, the packaged fish are transported to retailers via domestic distributors. The dairy supply chain proceeds similarly starting from cattle ranches, which produce milk that is transported daily to processing facilities with the help of logistics partners. Finally, the resulting milk cartons are transported to retailers by distributors.

We assume that the participants of each supply chain agree on using a blockchain as a distributed ledger of their transactions and to store the state of the various products. The goal is to facilitate end-to-end product provenance and cut down on costs. To this end, a permissioned blockchain ensures confidentiality, transaction durability, and acceptable performance [13]. The agreed-upon collaboration logic is implemented in the form of one or more smart contracts and deployed on the blockchain.

Due to overwhelming overhead, scalability issues, and low interest, it is, however, unrealistic to assume that the participants of all possible supply chains agree on the same blockchain platform. This means that multiple blockchain instances will likely co-exist to support different supply chains, e.g., for the seafood and the milk processing sectors. Furthermore, since many blockchain technologies exist with different trade-offs and guarantees [13], it is not possible to assume that all considered blockchain instances are of the same technology. For example, we assume that the seafood supply chain is managed by a private Ethereum instance that uses Proof-of-Authority (https://wiki.parity.io/Proof-of-Authority-Chains), and that the dairy supply chain is managed by an instance of the Hyperledger Fabric [14] blockchain. However, certain partners could be participants of multiple supply chains at the same time. For instance, the retail enterprise depicted on top in Figure 1 is connected to the two aforementioned blockchain instances and integrates their smart contracts into the Supply Chain Management System (SCMS) of its Enterprise Resource Planning (ERP) system. This is problematic and cumbersome, as it has to adapt to the heterogeneous specifics of these blockchain technologies, such as their APIs, data formats, supported protocols, etc.

Design of Integration Logic

SCIP [4] facilitates the uniform invocation, monitoring and querying of heterogeneous smart contracts so that blockchain participants can perform these operations without having to understand the technical details of the specific underlying blockchain technology. This is especially helpful for participants accessing multiple hetero-

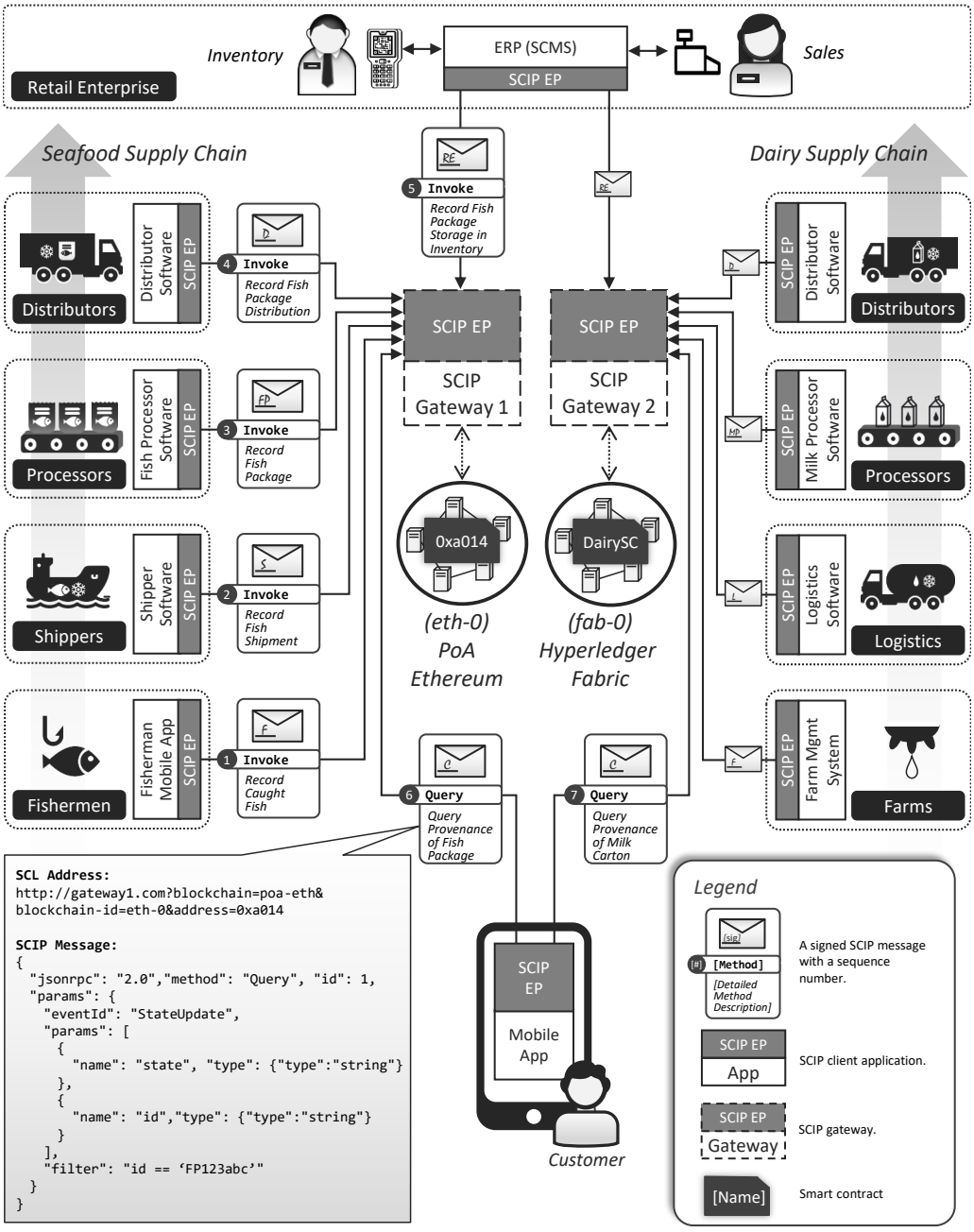

Figure 1. The supply chain integration scenario illustrated: a retail enterprise (top) integrates information from a seafood supply chain (left) and a dairy supply chain (right) to provide provenance information to its customers via a mobile app (bottom center). SCIP is used to interact with smart contracts through gateways, SCL to address them (center).

geneous blockchain instances at the same time, like the retail enterprise of the previous example.

However, since SCIP is not directly implemented by current blockchains, an external entity is needed that is connected to them on one hand and exposes a SCIP implementation to *SCIP client applications* on the other hand. We call this entity a *SCIP gateway* [4]. The gateway exchanges authenticated *SCIP messages* with the client applications connected to it, and translates them into technology-specific messages that are forwarded to the underlying blockchain instances via authorized nodes. Thus, it can be thought of as a uniform abstraction layer on top of heterogeneous blockchains.

There are multiple options as of which entity manages SCIP gateways; we plan to discuss these options in a future work. Here we adopt an option that aligns with the concept of Blockchain-as-a-Service (BCaaS [15]), as supported by providers like Amazon Web Services (https://aws.amazon.com/managed-blockchain), Upvest (https://upvest.co) or Kaleido (https://kaleido.io). Therefore, we assume that the participants of a supply chain entrust one or more partners among them to operate SCIP gateways that uniformly expose selected blockchain operations to client applications. Furthermore, client applications use SCL addresses to identify the smart contracts they wish to send SCIP messages to, and to define the specific blockchain instance; the SCL address up to the "?" implicitly and transparently to the client identifies the gateway (see Figure 1). Client applications become aware of the smart contracts' external interfaces and of how to correctly formulate SCIP messages by consulting their SCDL descriptors accessible, e.g., via a dedicated registry. For examples of SCDL descriptors, see online: https://github.com/floriandanielit/scdl#examples.

In the sample scenario in Figure 1, the participants of the *Seafood Supply Chain* have various client applications connected through *SCIP Gateway 1*. When a batch of fish is caught and moved along the supply chain, they issue smart contract function invocations in order to register it and update its state. SCIP supports the uniform invocation of smart contract functions via the `Invoke` method. Therefore, we see that the client applications of the *Fisherman* ❶, the *Shipper* ❷, the *Processor* ❸, and the *Distributor* ❹ send authenticated, i.e., digitally signed, SCIP `Invocation` messages to *SCIP Gateway 1*. Next, when a fish package is received by the *Retail Enterprise*, its ERP system triggers an additional `Invocation` message that announces its reception in the inventory ❺.

Later, the package is presented to *Customers* in one of the retailer's shopping centers. Interested buyers can then scan the barcode label on the package using a dedicated *Mobile App*. A SCIP `Query` message is then formulated and sent to *SCIP Gateway 1* ❻ (see the example SCIP message on the bottom left corner of Figure 1). The query requests previous occurrences of specific events that were triggered during the execution of smart contract functions. For example, the `Query` message shown in the figure is sent to the SCL of the smart contract responsible for handling product state changes of the seafood supply chain, which we assume has the address `0xa014`. The message retrieves the past occurrences of the event "StateUpdate" that emits two output parameters: "state" and "id". The "filter" field requires that the "id" value be equal to "FP123abc", which we assume corresponds to the identifier of the fish package the customer scanned. The result is a list of event occurrences that describe when each state change of the package happened and who caused it, i.e., it retrieves the package's provenance. This helps to convince the customer of the authenticity and quality of the packaged fish. Finally, similar queries can be directed to *SCIP Gateway 2* to retrieve the provenance of milk cartons ❼.

Implementation

We implemented a prototypical setup for the presented scenario, in which we included a minimal Hyperledger Fabric instance to manage the dairy supply chain and a simulated Ethereum blockchain instance, using Ganache (https://www.trufflesuite.com/ganache), to manage the seafood supply chain. We included two instances of JSON-RPC-based SCIP gateways with adapters for Ethereum and Hyperledger Fabric (code available at: https://github.com/ghareeb-falazi/BlockchainAccessLayer). We also deployed one smart contract on each blockchain to handle the collaboration logic via suitable functions and events. Finally, we implemented a simple web-based client application that allows the

various partners of the supply chains to control the lifecycle and custody of the circulated goods and allows customers to query the provenance of end packages.

The prototype is available on Github (https://github.com/ghareeb-falazi/SCIP-CaseStudy-2). It is containarized into multiple Docker (https://www.docker.com/) containers orchestrated using a combination of Docker Compose and Bash scripts.

DISCUSSION AND OUTLOOK

SCL, SCDL and SCIP lay the foundation for a service-oriented approach to the integration of heterogeneous blockchain smart contracts into generic software applications. They enable invoking smart contract functions, subscribing to new events, and querying for past ones. The specifications are open source[1], free and equipped with a reference implementation of a SCIP endpoint. Together, they advance the state of the art in BCaaS scenarios and bring the benefits of blockchains also to clients without specific blockchain competences. The discussed food supply chains scenario exemplified these benefits and showed how time-consuming and error-prone tasks, such as invoking smart contracts in different blockchains, can be achieved using uniform SCIP messages.

The approach is enabled by the sensible use of SCIP gateways and a delegated authentication scheme: while gateways create and sign blockchain transactions, SCIP requires clients to sign their messages allowing gateways to log (e.g., using a transaction) client signatures and corresponding transaction identifiers to enable the auditing of their work. Clients are identified and authenticated using standard OAuth 2.0 authorization tokens (https://oauth.net/2/).

While the described approach helps to solve the smart contract integration problem, it also opens up new challenges and research directions:

- Enterprise applications usually group the execution of different operations that semantically achieve a single collective goal into a *business transaction*. Based on their guarantees and properties, transactions implement different models, e.g., the ACID or the SAGA models. The question now is *which* kind of transactional models can be supported when multiple blockchain transactions of different blockchain platforms semantically form a larger business transaction. It is also unclear *how* such *cross-blockchain business transactions* can be executed, i.e., how the involved systems can coordinate their execution. In SCIP terms, this means that we need to be able to group different `Invocation` messages into cross-blockchain business transactions and define clear semantics for an atomic execution.
- The configuration, operation and management of *SCIP gateways* plays a key role in guaranteeing non-functional properties. For example, managing them by client systems does not require additional trust assumptions, but results in managerial overhead, whereas doing so by dedicated blockchain service providers introduces additional trust assumptions, but is management-free from the client system perspective. It is important to carefully analyse the resulting trade-offs, which also include aspects like performance, security, and scalability, and compile appropriate scenario-specific recommendations.
- Finally, SCIP provides limited query capabilities through the `Query` method. However, we identified the need for a more sophisticated and uniform *blockchain query language* to formulate expressive queries over events that took place in a blockchain system or its current state, regardless of the specific underlying technology. Similar to SQL in the domain of relational databases, such a query language could cater for a standard way to analyze blockchain data using conventional data analysis and visualization frameworks without having to implement individual plugins for each blockchain platform to be supported.

These challenges show how SCL, SCDL and SCIP may serve as a platform for the development of advanced, blockchain-oriented software engineering tools and methodologies. The case study discussed in this article shows how much these are also needed in practice.

[1]SCDL: https://github.com/floriandanielit/scdl
SCL: https://github.com/ghareeb-falazi/scl
SCIP: https://github.com/lampajr/scip

ACKNOWLEDGMENT

This research was partially funded by the Ministry of Science of Baden-Württemberg, Germany, for the doctoral program "Services Computing" and the DITAS project funded by the European Union, Horizon 2020, grant agreement RIA 731945.

REFERENCES

1. F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: current status, classification and open issues," *Telematics and Informatics*, vol. 36, pp. 55–81, 2019.
2. F. Daniel and L. Guida, "A service-oriented perspective on blockchain smart contracts," *IEEE Internet Computing*, vol. 23, no. 1, pp. 46–53, 2019.
3. A. Lamparelli, G. Falazi, U. Breitenbücher, F. Daniel, and F. Leymann, "Smart Contract Locator (SCL) and Smart Contract Description Language (SCDL)," in *Service-Oriented Computing – ICSOC 2019 Workshops*, 2019.
4. G. Falazi, U. Breitenbücher, F. Daniel, A. Lamparelli, F. Leymann, and V. Yussupov, "Smart Contract Invocation Protocol (SCIP): A Protocol for the Uniform Integration of Heterogeneous Blockchain Smart Contracts," in *32nd International Conference on Advanced Information Systems Engineering (CAiSE'20)*, 2020.
5. M. P. Papazoglou and D. Georgakopoulos, "Service-oriented computing," *Communications of the ACM*, vol. 46, no. 10, pp. 25–28, 2003.
6. T. Hardjono, A. Lipton, and A. Pentland, "Towards a Design Philosophy for Interoperable Blockchain Systems," *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1805.05934
7. S. Schmidt, M. Jung, T. Schmidt *et al.*, "Unibright-the unified framework for blockchain based business integration," White Paper, Apr. 2018.
8. M. Sporny and D. Longely, "The Web Ledger Protocol 1.0," W3C Blockchain Community Group, Tech. Rep., 2019. [Online]. Available: https://w3c.github.io/web-ledger/
9. X. Xu, C. Pautasso, L. Zhu, V. Gramoli, A. Ponomarev, A. B. Tran, and S. Chen, "The blockchain as a software connector," in *2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016*, 2016.
10. R. Yeung and W. M. Yee, "Food safety concern: Incorporating marketing strategies into consumer risk coping framework," *British Food Journal*, vol. 114, no. 1, pp. 40–53, 2012.
11. A. Awaysheh and R. D. Klassen, "The impact of supply chain structure on the use of supplier socially responsible practices," *International Journal of Operations and Production Management*, vol. 30, no. 12, pp. 1246–1268, 2010.
12. M. Montecchi, K. Plangger, and M. Etter, "It's real, trust me! Establishing supply chain provenance using blockchain," *Business Horizons*, vol. 62, no. 3, pp. 283–293, May 2019.
13. G. Falazi, V. Khinchi, U. Breitenbücher, and F. Leymann, "Transactional properties of permissioned blockchains," *SICS Software-Intensive Cyber-Physical Systems*, pp. 1–13, 2019.
14. E. Androulaki, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, A. Barger, S. W. Cocco, J. Yellick, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, and G. Laventman, "Hyperledger fabric," in *Proceedings of the Thirteenth EuroSys Conference on Computer Systems - EuroSys '18*. New York, New York, USA: ACM Press, 2018, pp. 1–15.
15. M. Samaniego and R. Deters, "Blockchain as a Service for IoT," in *2016 IEEE iThings / GreenCom / CPSCom / SmartData*. IEEE, 2016, pp. 433–436.

Ghareeb Falazi is a research associate and a PhD student at the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart. His research interests include distributed systems, and decentralized applications, focusing on the transactional properties of blockchain systems. He has received a M.Sc. degree in Computer Science from the University of Stuttgart in 2017. Contact him at ghareeb.falazi@iaas.uni-stuttgart.de.

Andrea Lamparelli is a M.Sc. student at Politecnico di Milano, Milan, Italy. His research interests include service-oriented computing and blockchains. Contact him at andrea.lamparelli@mail.polimi.it

Uwe Breitenbücher is a research staff member and postdoc at the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart, Germany. His research vision is to improve cloud application provisioning and application management by automating the application of management patterns. Uwe was part of the CloudCycle project, in which the OpenTOSCA Ecosystem was developed. His current research interests include cyber-physical systems, blockchains, and microservices. Contact him at uwe.breitenbuecher@iaas.uni-stuttgart.de.

Florian Daniel is an Associate Professor with Politecnico di Milano, Milan, Italy. His research interests include service-oriented computing, blockchain, business process management, and data science. He holds a Ph.D. in Information Technology from Politecnico di Milano. Contact him at florian.daniel@polimi.it.

Frank Leymann is a full professor of computer science and director of the Institute of Architecture of Application Systems (IAAS) at the University of Stuttgart, Germany. His research interests include service-oriented architectures and associated middleware, workflow- and business process management, cloud computing and associated systems management aspects, and patterns. Frank is co-author of more than 400 peer-reviewed papers, about 70 patents, and several industry standards. He is elected member of the Academy of Europe. Contact him at frank.leymann@iaas.uni-stuttgart.de.