Orchestration of Blockchain-based Digital Twins[*]

Marco Benedetti, Marco Favorito and Matteo Nardelli

Banca d'Italia

Abstract
Digital Twins (DTs) provide virtual copies of physical systems, hence representing a fundamental building block for Smart Manufacturing. The advent of blockchain offered new features for DTs, which can be exposed as services through smart contracts whose interaction can be autonomous and event-based. Although the literature highlights the importance of DTs collaboration, approaches to the automatic composition of DTs in a blockchain-based scenario have only recently been investigated. In this paper, we present a novel framework for DT service composition, where the blockchain plays a fundamental role in data management, process design and process execution.

Keywords
Digital Twins, Blockchain, Service Composition

1. Introduction

The Fourth Industrial Revolution, also known as "Industry 4.0" [1] is a paradigm shift in how industrial activities are conceived and executed, characterized by increased automation, digitalization and inter-connectivity of industrial systems. In this context, Smart Factories [2] refer to factories equipped with sensors, actors, and autonomous components, and made of Cyber-Physical Systems (CPS). In CPS, the physical level and the digital level merge together, both at the production process level as well as the product level. Digital Twins (DTs) [3, 4] are a fundamental building block for Smart Manufacturing. They are virtual copies of machines or systems that, driven by data collected from sensors in real time, mirror almost every facet of a product, process, or service. They allow drawing intelligent conclusions from data by identifying faults, recommending precautionary measures ahead of critical events, and, more generally, allowing for better-informed decision-making. The use of DTs in the industry is already a reality, as also demonstrated by ISO 23247 standards on DT creation [5]. Among the first industrial applications, NASA used digital copies to monitor the status of its spacecrafts [6], General Electric used them to track the operations of wind turbines [7], and Siemens applied DTs for the power system and wastewater plant in Finland [8].

Recent research efforts look into novel architectures and techniques to orchestrate and compose DTs in order to increase automation of industrial processes, e.g., [9, 10, 11, 12, 13, 14, 15, 16, 17].

DLT'23: 5th Distributed Ledger Technology Workshop, May 25–26, 2023, Bologna, Italy

[*] All views are those of the authors and do not necessarily reflect the position of Bank of Italy.

marco.benedetti@bancaditalia.it (M. Benedetti); marco.favorito@bancaditalia.it (M. Favorito); matteo.nardelli@bancaditalia.it (M. Nardelli)
0000-0001-9566-3576 (M. Favorito); 0000-0002-9519-9387 (M. Nardelli)

In particular, instead of designing the entire manufacturing process in a traditional way, inspired by the *Web Service Composition* literature [18], Catarci et al. [11] envision a smart manufacturing architecture where human designers can specify a target service and take advantage of automatic composition to realize the corresponding physical processes. DTs provide stateful services, wrapping the functionalities of physical systems and tasks of human operators. The automatic composition allows expressing the target service as a composition of the DTs' services, by providing a solution that schedules the target service actions to the available services provided by DTs. Recently, there has been also considerable interest in applying blockchain technologies to smart factories [19] and in DT scenarios [20], to address the lack of trust among stakeholders and traceability of data and industrial processes.

Nonetheless, previous works on blockchains and DTs do not focus on the automation of the manufacturing process and how to orchestrate DTs and the available services. To the best of our knowledge, the use of a service composition approach in a blockchain-based DTs scenario has not been explored so far. Inspired by [11], in this paper, we propose a framework for orchestrating DTs where the blockchain plays a fundamental role in (i) data management, both of the manufacturing process and of the state of the DTs, (ii) process design based on the Roman Model service composition technique [21, 22], and (iii) process execution by means of automated event-based communication between smart contracts controlling the available DTs. We imagine such a framework to be very useful in scenarios where there is the need for a shared database with an objective immutable log on relevant facets of the execution of the business process, and multiple parties involved with conflicting interests and trust issues [23].

2. Related Work

Industry 4.0 and Blockchain. The research community on business process management (BPM) started to explore how blockchains can help rethink processes and their management. Mendling et al. [24] propose a broad analysis of challenges and opportunities. In [25], blockchains are used to improve information transparency and decentralization in a cloud manufacturing scenario. With a special emphasis on Industry 4.0, Viriyasitavat et al. [26] suggest using the blockchain to trace non-functional requirements of services, execution requests, and feedback; the authors aim to build trust among services while reducing intermediation costs. Notably, Weber et al. [27] solve the trust problem in collaborative process execution using blockchain, which can either act as a choreography monitor, by storing the process execution status, or as an active mediator among the participants, by coordinating the collaborative process execution. Our contribution builds upon this idea.

DTs and Blockchain. A *Digital Twin* (DT) is a synchronized virtual replica of an underlying product, process, or service, which can be used to analyze, manage, and optimize all operations on its real-world implementation [28, 29, 30, 31]. *Blockchain-based DTs* are receiving an ever-growing interest, as they target several key challenges in DTs scenarios such as disparate data silos, untrustworthy data dissemination, traceability, and the need for predictive maintenance [32]. When deployed into a DT environment, smart contracts can be used to track data sharing, (e.g., [33, 34]), to trace the twins' life cycle (e.g., [35, 36]), to store authorization information for all involved parties (e.g., [37, 38]), or to automate event-based interaction among machines (e.g., [39]). Notably, Angrish et al. [39] present a case study for Blockchain in manufacturing, where the blockchain enables a decentralized and event-based interaction among machines and humans. Although this work explores principles we adopt in our work, it does not investigate service composition. Pittaras et al. [13] uses smart contract to expose DTs actions, which can be executed upon the payment of a price expressed in tokens. Composition is not the scope of their work. In this paper, we consider physical machines and real assets having their own DTs with an associated smart contract on the blockchain. Therefore, we focus on smart contracts to provide automation capabilities and to coordinate the execution of multiple DTs (thus enabling the implementation of complex processes or services). Differently from EtherTwin [38], we store "sensor data"—aggregated according to a finite-state machine (FSM) specification—on-chain rather than on off-chain.

Service composition. The proliferation of vendors and the increase in the complexity of manufacturing processes poses several challenges in achieving a modular and flexible implementation of such processes. As pointed out in [11], this view shares some analogies with the notion of *Web service composition.* The problem of service composition has been largely considered in the literature, starting from seminal manual approaches (e.g., [40, 41, 42]), which mainly focused on modeling issues and on automated discovery using rich ontologies, to automatic approaches based on planning (e.g., [43, 44]), knowledge representation (e.g., [45]), or automated synthesis (e.g., [21]). We refer the interested reader to existing surveys for further details, e.g., [18, 46, 22].

Composition of DTs. Only recently, service composition has been proposed in a DT setting. Sahal et al. [47] show the key importance of DTs collaboration for achieving higher quality production. Besides introducing a lightweight framework of DTs collaboration, the authors present several real use cases. Wilhelm et al. [48] cover this topic with a systematic literature review. Inspired by the Roman Model [21, 49], Catarci et al. [11] envision an architecture human designers take advantage of automatic composition to realize the physical manufacturing processes. Leotta and Mecella [50] propose a conceptual architecture for the dynamic composition of DTs. Also Pernici et al. [12] suggest combining Service Oriented Architectures with CPS; nonetheless, they focus on exchanging data in a dynamic and adaptive way. Differently from our approach, these works do not consider blockchain for exposing DTs functionalities and for supporting event-based interaction.

3. Background

3.1. Blockchains and Smart Contracts

A *blockchain* is a distributed database technology that builds on a tamper-proof list of timestamped transaction records, organized as a sequence (or, chain) of immutable blocks. The chain is distributed over a peer-to-peer network, where every node maintains the latest version of it, acting as a state-machine replication mechanism. Since every node can create blocks in a peer-to-peer network, they have to agree on the next block to add to the chain. This allows implementing a *trustless* system. A *smart contract* [51] is a program living in the blockchain, deployed with a specific type of transaction, and whose execution is triggered by means of transactions addressed to it. Depending on the expressiveness of the programming language, a smart contract can implement arbitrary business logic (e.g., conditional money transfer).

The smart contract code is deterministic and relies on a closed-world assumption: the only information that is stored on the blockchain is available in the run-time environment.

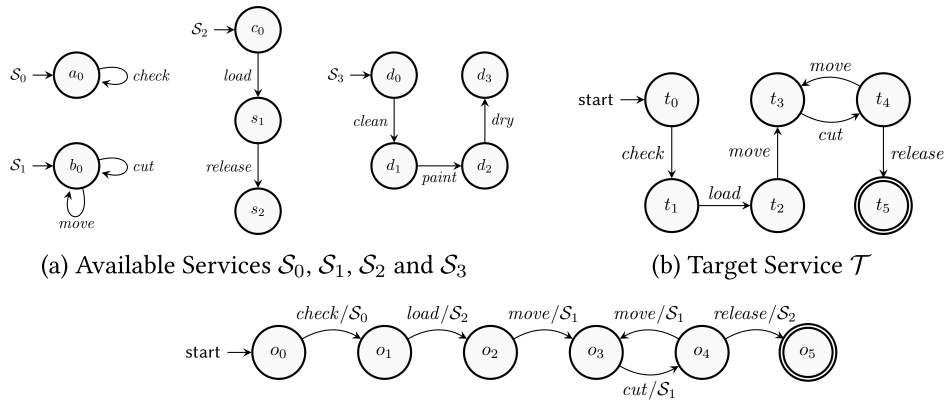
3.2. The Roman Model

In the Roman model [21], the composition is as follows. Each available service is modeled as a finite state machine (FSM): at each state, the service offers a certain set of actions, where each action changes the state of the service in some way. Formally, a *service* is defined as a tuple $\mathcal{S} = \langle \Sigma, \sigma_0, F, A, \delta \rangle$ where: Σ is the finite set of service states, $\sigma_0 \in \Sigma$ is the initial state, $F \subseteq \Sigma$ is the set of the service final states, A is the finite set of service's actions, $\delta \subseteq \Sigma \times A \rightarrow \Sigma$ is the service's deterministic and partial transition function. The designer is interested in generating a new *target service* $\mathcal{T}$ from the set of existing services, which is specified using an FSM, too. The available services are grouped into *community* $\mathcal{C} = \{\mathcal{S}_1, \ldots, \mathcal{S}_n\}$, where they share a common set of actions A, i.e., the actions of the community. An *orchestrator* is an entity able of scheduling services on a step-by-step basis and of coordinating the community services to mimic the behavior of the target service. Formally, an orchestrator for a community $\mathcal{C}$ is a partial function $\gamma : \Sigma_1 \times \cdots \times \Sigma_n \times A \rightarrow \{1 \ldots n\}$, with Σ_i the set of states of $\mathcal{S}_i \in \mathcal{C}$, $i \in \{1 \ldots n\}$. Intuitively, γ is a decision maker that keeps track of the evolution of the services in the community up to a certain point and, in response to an incoming action request, returns the index of the service where to dispatch the request. The composition problem amounts to see whether the target service is *realizable*, i.e., there exists an orchestrator that properly orchestrates the work of the component services for all the possible behaviors of the target service. If an orchestrator γ satisfies such condition, we say that γ *realizes* $\mathcal{T}$. In this case, γ is also called a *composition* of $\mathcal{T}$. Thus, a new service $\mathcal{T}$ is synthesized using existing services.

4. Motivating Example

To better motivate the vision behind the orchestration of blockchain-based DTs, we consider the example of a *cutting machine* used to craft 2D objects to be later assembled. A cutting machine is usually an expensive equipment that a factory can *buy or rent* to provide its designers with a tool enabling rapid prototyping. An *authorized* designer could submit a newly conceived chair project to the cutting machine, which can - in turn - craft the individual components of the chair from sheets of wood. The cutting process is not for free: e.g., the blade wears out during cutting, raw materials may be out of stock. Monitoring the machine and its status is of utmost importance, e.g., to perform predictive maintenance, avoid wastage, and optimize the production process. The cutting machine can perform different kinds of tasks, so a *programmable software* usually enables its operations. It can account for constraints on the blade trajectory, cutting size, or waste optimization. Leveraging the smart manufacturing principles, the cutting machine could be remotely controlled, exploiting the key principles introduced by DTs. This enables the cutting machine to receive the processing logic from *remote* locations.

A complex manufacturing process usually requires combining and coordinating the execution of multiple machines and actuators. DTs make it simple to combine and control the execution of multiple physical devices, thus moving a step closer to the idea of cloud manufacturing [52, 53]. However, besides improving flexibility, this also raises concerns regarding trust, traceability,



$\mathcal{S}_0$ a_0 check
$\mathcal{S}_1$ b_0 cut move
$\mathcal{S}_2$ c_0 load s_1 release s_2
$\mathcal{S}_3$ d_0 clean d_1 paint d_2 dry d_3
start t_0 check t_1 load t_2 move t_3 cut move t_4 release t_5
start o_0 check/$\mathcal{S}_0$ o_1 load/$\mathcal{S}_2$ o_2 move/$\mathcal{S}_1$ o_3 move/$\mathcal{S}_1$ cut/$\mathcal{S}_1$ o_4 release/$\mathcal{S}_2$ o_5

(a) Available Services $\mathcal{S}_0$, $\mathcal{S}_1$, $\mathcal{S}_2$ and $\mathcal{S}_3$

(b) Target Service $\mathcal{T}$

(c) Orchestration

Figure 1: Example of service orchestration: Fig. 1a shows the available services exposed by the DTs. Fig. 1b shows the target service, whereas Fig.1c shows a possible orchestration for it.

and accountability. A blockchain can help to overcome these issues, by providing notarization functionalities, data tracing, and tracking the actions performed on each DT (and, in turn, on the physical machine). The blockchain enables the equipment (through its DT) to expose its functions by means of smart contracts, which can grant access rights, list the available operations, and enqueue requests for actions. Then, anyone accessing the smart contract can use such a functionality by sending action requests. Ultimately, the DT will be responsible for reading requests and enacting them on the physical device.

We consider DTs that use smart contracts to define the *services* (i.e., functionality) offered by the physical machines. We reference them as *available services*. For example, Figure 1a reports four of them: $\mathcal{S}_0$ can checks the machine state; $\mathcal{S}_1$ allows to program the blaze and perform the cut; $\mathcal{S}_2$ enables loading and releasing raw material from the cutting machine; $\mathcal{S}_3$ is able of cleaning, paint, and dry the processed material within the machine. To use and compose these services, a user can define a so-called *target service*. This service does not exist in reality but can be obtained by using the available services. Figure 1b shows a service $\mathcal{T}$ defining a simple cutting process. The framework we propose in Section 5 selects among the available services and orchestrates the execution of actions on them aiming to implement the target service. Figure 1c also shows the resulting *orchestration* for $\mathcal{T}$ using $\mathcal{S}_0$, $\mathcal{S}_1$, and $\mathcal{S}_2$.

5. Framework

This section describes our framework for the orchestration of blockchain-based DTs. It can be useful whenever, in a business process implementation, there is lack-of-trust among its stakeholders. Therefore, full traceability and accountability is required to resolve disputes, or to perform quality-of-service (QoS) tasks such as predictive maintenance or process optimization.

In this work, we imagine the framework in the context of a blockchain-based Industry 4.0 scenario, where the smart factory is endowed with a private permissioned smart-contract-enabled blockchain layer used for data sharing and as a communication medium among its

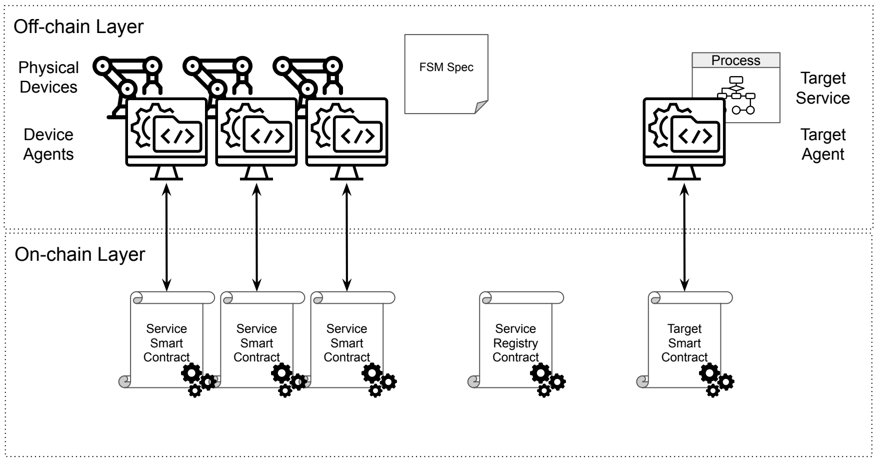


Figure 2: High-level architecture of the proposed framework.

subsystems. The available machines and real assets have their own DT with an associated smart contract on the blockchain (*à la* EtherTwin [38]). Our aim is to implement a target manufacturing process by composing the existing machines; that is, to find an orchestrator that realizes the process, as in the Roman model, and to deploy, *as a smart contract*, the process implementation in the smart factory. The resulting smart contract acts as a *mediator*, driving the process according to the process specification and the orchestrator's logic. This smart contract is conceptually very similar to the mediator contract used in [27]. Observe that our approach can be generalized also to other scenarios.

5.1. High-level Overview

A high-level overview of the architecture is shown in Figure 2. The system is divided into two layers: an *off-chain layer*, where the industrial process takes place and the physical assets reside, and an *on-chain layer*, which serves as a trustless notarization layer and as a mean of machine-to-machine communication between off-chain software components. Dedicated software *agents* take care of synchronization and consistency resolution between the physical devices (off-chain) and their virtual representatives (on-chain).

5.1.1. Off-chain Layer

In the off-chain layer, we have physical devices and the target services to implement, together with the related agents (i.e., the device and target agents).

Physical devices are specialized machines available in the smart factory. Such devices can execute functions and tasks useful for the realization of a target industrial process. We assume their behavior can be abstracted into FSM specifications. Each physical device has its own DT, acting as unique representative in the digital space. The *device agent* oversees the relationship between the physical device and its DT. As such, it has the capability of requesting the execution of tasks to its physical device, and retrieve the state of the machine from sensor data. In particular, it is able to determine whether a requested action has been successfully completed. Crucially, the device agent is the *off-chain representative* of its physical device, to be compared with the *service smart contract* (defined below), which is the *on-chain representative* of the device. Finally, the device agent is endowed with a blockchain identity (e.g., an Ethereum address), and therefore is able to both read/write data from/to the blockchain.

The *target service* is the industrial process we wish to accomplish. This can be achieved by opportunely instructing the device agents to ask for the execution of specific actions to their physical devices. We assume the dynamics of the target service can be abstracted into a FSM, where the labels of the transitions are associated with actions that can be executed by the available machines. Note that the target service is not (necessarily) associated with a physical entity but has to be intended as a specification of an industrial process that can be implemented using the available services. Nevertheless, the outcome of the target service can be a physical asset, e.g., a product that passes through an assembly line. We allow the target service specification to represent different histories of execution. At runtime, the actual evolution of the target service is determined by its stakeholders. The *target agent* is the DT for the target service. It is analogous to the role of the device agent for physical devices and acts as a bridge between the target service execution and the on-chain layer. In particular, it is the off-chain representative of the target service, whereas the *target smart contract* (defined below) is the on-chain representative of the process.

We omit the details about the interaction between the device agents and the physical agent, as the actual technology stack being used can vary a lot among concrete instances of the proposed framework.

5.1.2. On-chain Layer

The components that live in the on-chain layer are the service registry as well as the service and the target smart contracts.

The *service registry contract* serves as an on-chain registry of the service (contracts) available in the system. It accepts service registration requests from device agents (registerService()) and allows retrieving the list of available services (getServices()). We assume the contract is already deployed on the blockchain, and it is known to all stakeholders.

The *service smart contract* is the on-chain representative of a physical device. It is generated and deployed by the device agent, which is also the owner of the contract. The device agent can update the state of the contract so to reflect the actual off-chain state of the machine on the on-chain layer. The smart contract can notify the device agent about action requests coming from the target smart contract. The communication details are postponed to Section 5.1.3.

The *target smart contract* is the on-chain representative of the target agent. It is generated and deployed by the target agent, and it contains the solution to the composition problem: it implements an orchestrator that realizes the target service with the available services. As such, it can accept action requests from the target agent, and dispatch the request to the service contract determined by the orchestration policy. This contract also keeps track of the system state and the state of the target service.

Intuitively, by means of the transactions addressed to the above-mentioned contracts, the on-chain layer traces the target service life-cycle: the recording of the state of physical devices (through their DTs), the creation of a target service instance, and the execution of the target service by dispatching requests to the chosen available services.

5.1.3. Communication Patterns

In this section, we briefly explain the communication between the system participants.
Off-chain/Off-chain. In our abstract framework, we do not consider direct communication between off-chain components; except for the communication between physical devices and their DTs. However, in our framework we are not interested in modeling the actual working of the machines. For the target service, the pair physical device/device agent appears as a unique participant of the system.
On-chain/On-chain. Communication between smart contracts happens through function calls. The contracts are referenced by their address. Being a purely reactive component, the smart contract cannot exhibit proactive behavior: the first function call is always triggered by either a device agent or the target agent.
Off-chain/On-chain. The communication between off-chain and on-chain layers is bidirectional, although implemented in different ways. The *off-chain/on-chain* communication happens through the submission of blockchain transactions (by device or target agents) and addressed to their twin contract supposed to receive the message. The *on-chain/off-chain* communication is built on the concept of blockchain *events*: the off-chain component keeps listening events emitted by the blockchain[1]. The actual communication protocol employed by the participant will be discussed in the next sections.

5.2. Workflow

In this section, we describe the workflow phases that bring the completion of the industrial process, starting from the *registration phase*, where the services advertise themselves on-chain as available; the *composition phase*, where the target agent retrieves the available services, computes a solution to the composition problem, and deploys the computed orchestrator as a smart contract (the target smart contract); the *setup phase*, where the target agent checks and reserves the availability of the chosen services; and the *execution phase*, where the actual target service gets executed and each requested action dispatched to the chosen service. An overview of the setup and the execution phases is shown in Figure 3. The workflow assumes the blockchain up and running, with an instance of the service registry contract already deployed.

5.2.1. Registration

The device agents register themselves to the service registry, aiming to be discoverable by candidate target services. First, the device agent uses the FSM specification of the physical device to generate a smart contract that keeps track of the state changes of the machine according to its specification. In our prototype, we use a JSON format as specification language and Solidity as smart contract programming language. The idea to generate smart contract code from a

[1]For example, in the Ethereum ecosystem, a blockchain node serves a JSON-RPC API over WebSockets that allows to subscribe to specific events, emitted by smart contract code, using a publisher/subscriber pattern: https://ethereum.org/en/developers/tutorials/using-websockets/#subscription-api

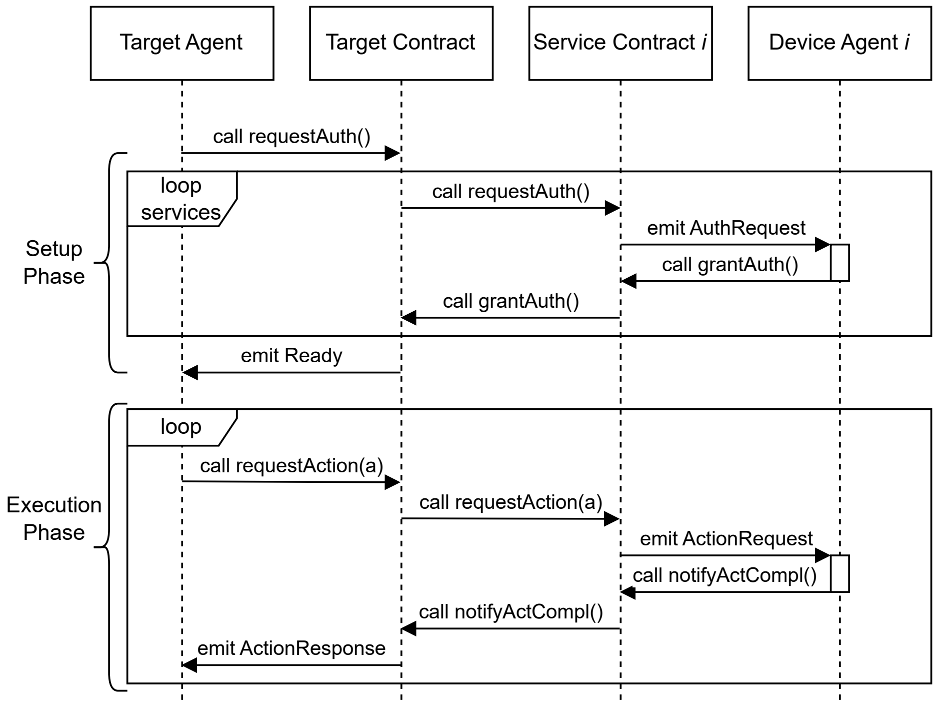


Figure 3: Sequence diagram showing the interactions between the system participants for the setup and the execution phases.

FSM specification is inspired by the FSolidM tool [54]. The generated contract code is then compiled and deployed as a service smart contract. Then, the service smart contract is registered to the service registry contract, using the registerServicefunction. Finally, the device agent connects with the blockchain node and listens for events emitted by the service contract (see Section 5.1.3).

5.2.2. Composition

The target agent retrieves from the service registry contract the available devices and their FSM specifications, using the function getServices. Once downloaded all the specifications, the target agent uses them, together with the FSM specification of the target service, to compute a service composition. In case the target is realizable with the community of available services, the composition algorithm outputs an orchestrator (i.e., a solution to the composition problem). The actual algorithm can be, e.g., the *ND-Simulation* algorithm [22, Algorithm 1]). The orchestrator is used to generate the source code of the target smart contract, which is then compiled and deployed on the blockchain. Finally, the target agent connects to the blockchain waiting for events coming from the target contract.

5.2.3. Setup

Once both the service contracts and the target contract are deployed, the target agent probes the availability of the chosen services, to reserve their usage for the execution of the process. To do so, the target agent calls the contract function requestAuthorization() of the target contract. The function iterates over all the chosen services and forwards to each of them the availability request by calling the function requestAuthorization(). If the service contract is already assigned to another target contract, then the request will fail. Otherwise, the request is stored in the contract state as "pending", and the contract emits an event notifying the device agent about the new availability request. The device agent can evaluate the request and apply its own authorization policy, i.e., decide whether to grant the authorization to the target service to use the device it represents. In case of success, the device agent grants the authorization by calling the method grantAuthorization() of the service contract. In turn, the service contract notifies the target contract whose request was pending by calling its function grantAuthorization(). When all services granted the authorization for being used, the target contract emits the event Ready to notify the target agent that the execution of the target service can begin.

5.2.4. Execution

Once the target agent obtained the authorization for using the available services, it starts requesting the execution of actions. To do so, it asks the target contract to dispatch the request of the needed action to the chosen service according to the orchestration policy computed in the composition phase. The target agent leverage the target contract to request action aiming to improve traceability and accountability. Moreover, this approach has two further benefits. First, it readily mirrors the case where the target agent is a DT of real industrial service, whose actions cannot be completely automated. Second, it overcomes the limitation of current blockchains that do not support fully autonomous and event-based smart contracts. The target agent calls the function requestAction(action) of the target contract, which in turn calls the requestAction(action) function of the chosen service contract. This function stores the action request in the contract state as "pending" and emits the ActionRequest event to notify the device agent that a new action request is pending. The device agent receives the event and processes the action request by instructing the controlled physical device to execute it. When the action is completed, the device agent calls notifyActionCompleted on its service contract, which notifies completion on the target contract that, ultimately, emits the ActionResponse event for the target agent. This sequence of events allows the device agent to notify the target agent that the action previously requested has been fulfilled successfully. The target agent can then request a new action to the target contract.

6. Discussion

This paper represents a preliminary work aimed to open future research directions in the area of automatic composition of smart contract. Although such autonomous capabilities can be of broad interest for generic blockchain-based applications, we believe that they could be particularly relevant for industrial settings where automation, integration, and accountability are of utmost importance. An example is discussed in Section 4.

To this end, we designed a very general framework (Section 5), which is well suited to host different and more advanced smart contract composition techniques. As proof of concept, we resort on the so-called Roman model, described in Section 3.2. Nonetheless, we envision that novel approaches can be defined, which compose smart contracts aiming to address functional and non-functional requirements. As suggested in Section 2, the literature on service composition can provide insights on methodologies suited for this task.

Before concluding, we share some of the lessons learned on modeling smart contracts as FSMs in the context of the Roman model. We believe that the classical Roman model has some limitations in how complex smart contracts can be modeled, especially regarding state variables with infinite (or very large, e.g., 32-bit integers) domains, or lack of message-based communication modeling widely used in smart contracts. To address the above issues, it might be interesting to explore the *Colombo* framework for web service composition [55]. On the other hand, the simplicity of the Roman model can still be useful whenever a FSM can correctly capture the behaviour of the smart contract, and it can be easily extracted (either from the contract code or its specification).

7. Conclusion

This paper explores, for the first time, the use of a service composition approach in a blockchain-based DT scenario. It proposes a framework to design systems that take advantage of a blockchain as an immutable log of events, as a data-sharing platform, and as an enabler of automated machine scheduling by exploiting service composition techniques.

Although it is a preliminary work, we argue that it lays the foundation for further developments in the automated composition of blockchain-based DTs. We foresee several interesting research directions, including:

- **Error Handling.** An extension of the proposed framework that is able to capture exception handling procedures [56]: if the target service terminates before a terminal state has been reached, work done so far might have to be explicitly undone (by rollbacks or compensation actions). Other exceptions to the "happy path" that might occur in a real-world scenario are the denial or the revocation of the authorization for the target process to use some device, the failure of action execution, and the use of timeouts to detect unresponsive device agents.
- **Advanced Composition.** Different service composition approaches, or modeling formalisms, can help to overcome the limitations of the Roman Model, such as the limited expressive power of the FSM model for services or the inability to model the stochastic non-determinism of service dynamics. Markov Decision Processes could be used to model stochastic services in place of FSMs (e.g., as [57, 15, 14] do). Composition on non-functional requirements could be considered as well. The orchestration policy can be computed, or updated at run-time, by taking into account QoS metrics. Past performance of the available services can be inferred from the events stored in the blockchain.
- **Use cases.** Exploring new use cases in Industry 4.0, e.g., coming from the supply chain industry (as in [27]), or in completely different domains where the blockchain-based

service composition might be an effective tool, e.g., in the Web3 ecosystem [58].

References

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, M. Hoffmann, Industry 4.0, Business & information systems engineering 6 (2014) 239–242.

[2] D. Lucke, C. Constantinescu, E. Westkämper, Smart factory-a step towards the next generation of manufacturing, in: Manufacturing systems and technologies for the new frontier, Springer, 2008, pp. 115–118.

[3] Y. Lu, C. Liu, K. I. Wang, H. Huang, X. Xu, Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues, Robotics Comput. Integr. Manuf. 61 (2020) 101837.

[4] F. Tao, Q. Qi, Make more digital twins, 2019.

[5] ISO/DIS-23247-1, Automation systems and integration–digital twin framework for manufacturing–part 1: Overview and general principles, 2020.

[6] E. Glaessgen, D. Stargel, The digital twin paradigm for future nasa and us air force vehicles, in: 53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA, 2012, p. 1818.

[7] A. M. Lund, K. Mochel, J.-W. Lin, R. Onetto, J. Srinivasan, P. Gregg, J. E. Bergman, K. D. Hartling, A. Ahmed, S. Chotai, et al., Digital twin interface for operating wind farms, 2018. US Patent 9,995,278.

[8] M. Siemens, For a digital twin of the grid: Siemens solution enables a single digital grid model of the finnish power system (2017). URL: https://assets.new.siemens.com/siemens/assets/api/uuid:09c20834-4ed4-49d8-923d-ebcc541cab37/inno2017-digitaltwin-e.pdf.

[9] F. Tao, M. Zhang, Digital twin shop-floor: A new shop-floor paradigm towards smart manufacturing, IEEE Access 5 (2017) 20418–20427.

[10] X. V. Wang, L. Wang, Digital twin-based WEEE recycling, recovery and remanufacturing in the background of industry 4.0, Int. J. Prod. Res. 57 (2019) 3892–3902.

[11] T. Catarci, D. Firmani, F. Leotta, F. Mandreoli, M. Mecella, F. Sapio, A conceptual architecture and model for smart manufacturing relying on service-based digital twins, in: ICWS, IEEE, 2019, pp. 229–236.

[12] B. Pernici, P. Plebani, M. Mecella, F. Leotta, F. Mandreoli, R. Martoglia, G. Cabri, et al., Agilechains: agile supply chains through smart digital twins, in: Proc. Eur. Saf. Reliab. Conf. Probab. Saf. Assess. Manag. Conf., Proceedings of the 30th European Safety and Reliability Conference and 15th Probabilistic Safety Assessment and Management Conference, Research Publishing Singapore, 2020, pp. 2678–2684.

[13] I. Pittaras, N. Fotiou, C. Karapapas, V. A. Siris, G. C. Polyzos, Secure, mass web of things actuation using smart contracts-based digital twins, in: 2022 IEEE Symposium on Computers and Communications (ISCC), 2022, pp. 1–6. doi:10.1109/ISCC55528.2022.9912991.

[14] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, L. Silo, Digital twins composition via

markov decision processes, in: ITBPM@BPM, volume 2952 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 44–49.

[15] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, L. Silo, Modeling resilient cyber-physical processes and their composition from digital twins via markov decision processes, in: PMAI@IJCAI, volume 3310 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 101–104.

[16] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, L. Silo, Digital twins composition in smart manufacturing via Markov decision processes, Computers in Industry 149 (2023) 103916. URL: https://www.sciencedirect.com/science/article/pii/S0166361523000660. doi:https://doi.org/10.1016/j.compind.2023.103916.

[17] G. De Giacomo, M. Favorito, F. Leotta, M. Mecella, F. Monti, L. Silo, Aida: A tool for resiliency in smart manufacturing, in: CAiSE Forum, Lecture Notes in Business Information Processing, Springer, 2023.

[18] A. L. Lemos, F. Daniel, B. Benatallah, Web service composition: a survey of techniques and tools, ACM Computing Surveys (CSUR) 48 (2015) 1–41.

[19] J. Leng, S. Ye, M. Zhou, J. L. Zhao, Q. Liu, W. Guo, W. Cao, L. Fu, Blockchain-secured smart manufacturing in industry 4.0: A survey, IEEE Trans. Syst. Man Cybern. Syst. 51 (2021) 237–252.

[20] S. Suhail, R. Hussain, R. Jurdak, C. S. Hong, Trustworthy digital twins in the industrial internet of things with blockchain, IEEE Internet Comput. 26 (2022) 58–67.

[21] D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, M. Mecella, Automatic composition of e-services that export their behavior, in: International conference on service-oriented computing, Springer, 2003, pp. 43–58.

[22] G. De Giacomo, M. Mecella, F. Patrizi, Automated service composition based on behaviors: The roman model, in: Web Services Foundations, Springer, 2014, pp. 189–214.

[23] A. B. Pedersen, M. Risius, R. Beck, et al., A ten-step decision path to determine when to use blockchain technologies, MIS Quarterly Executive 18 (2019) 99–115.

[24] J. Mendling, I. Weber, W. M. P. van der Aalst, J. vom Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar, A. Gal, L. García-Bañuelos, G. Governatori, R. Hull, M. L. Rosa, H. Leopold, F. Leymann, J. Recker, M. Reichert, H. A. Reijers, S. Rinderle-Ma, A. Solti, M. Rosemann, S. Schulte, M. P. Singh, T. Slaats, M. Staples, B. Weber, M. Weidlich, M. Weske, X. Xu, L. Zhu, Blockchains for business process management - challenges and opportunities, ACM Trans. Manag. Inf. Syst. 9 (2018) 4:1–4:16.

[25] C. Yu, L. Zhang, W. Zhao, S. Zhang, A blockchain-based service composition architecture in cloud manufacturing, International Journal of Computer Integrated Manufacturing 33 (2020) 701–715.

[26] W. Viriyasitavat, L. Da Xu, Z. Bi, A. Sapsomboon, Blockchain-based business process management (bpm) framework for service composition in industry 4.0, Journal of Intelligent Manufacturing 31 (2020) 1737–1748.

[27] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, J. Mendling, Untrusted business process monitoring and execution using blockchain, in: BPM, volume 9850 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 329–347.

[28] F. Tao, J. Cheng, Q. Qi, M. Zhang, H. Zhang, F. Sui, Digital twin-driven product design, manufacturing and service with big data, Int J Adv Manuf Technol 94 (2018) 3563–3576.

doi:10.1007/s00170-017-0233-1.

[29] F. Tao, Q. Qi, L. Wang, A. Nee, Digital twins and cyber–physical systems toward smart manufacturing and industry 4.0: Correlation and comparison, Engineering 5 (2019) 653–661. doi:https://doi.org/10.1016/j.eng.2019.01.014.

[30] A. Rossmann, D. Hertweck, Digital twins: a meta-review on their conceptualization, application, and reference architecture, in: Proceedings of the 55th Hawaii International Conference on System Sciences (HICSS 2022), University of Hawai'i at Manoa, 2022, pp. 4518–4527.

[31] K. Y. H. Lim, P. Zheng, C.-H. Chen, A state-of-the-art survey of digital twin: techniques, engineering product lifecycle management and business innovation perspectives, Journal of Intelligent Manufacturing 31 (2020) 1313–1337. doi:10.1007/s10845-019-01512-w.

[32] S. Suhail, R. Hussain, R. Jurdak, A. Oracevic, K. Salah, C. S. Hong, R. Matulevičius, Blockchain-based digital twins: Research trends, issues, and future challenges, ACM Comput. Surv. 54 (2022).

[33] S. Huang, G. Wang, Y. Yan, X. Fang, Blockchain-based data management for digital twin of product, Journal of Manufacturing Systems 54 (2020) 361–371.

[34] J. J. Hunhevicz, M. Motie, D. M. Hall, Digital building twins and blockchain for performance-based (smart) contracts, Automation in Construction 133 (2022) 103981. doi:https://doi.org/10.1016/j.autcon.2021.103981.

[35] C. P. Nielsen, E. R. da Silva, F. Yu, Digital twins and blockchain – proof of concept, Procedia CIRP 93 (2020) 251–255. doi:https://doi.org/10.1016/j.procir.2020.04.104.

[36] H. R. Hasan, K. Salah, R. Jayaraman, M. Omar, I. Yaqoob, S. Pesic, T. Taylor, D. Boscovic, A blockchain-based approach for the creation of digital twins, IEEE Access 8 (2020) 34113–34126.

[37] M. Dietz, B. Putz, G. Pernul, A distributed ledger approach to digital twin secure data sharing, in: IFIP Annual Conference on Data and Applications Security and Privacy, Springer, 2019, pp. 281–300.

[38] B. Putz, M. Dietz, P. Empl, G. Pernul, Ethertwin: Blockchain-based secure digital twin information management, Inf. Process. Manag. 58 (2021) 102425.

[39] A. Angrish, B. Craver, M. Hasan, B. Starly, A case study for blockchain in manufacturing:"fabrec": A prototype for peer-to-peer network of manufacturing nodes, Procedia Manufacturing 26 (2018) 1180–1192.

[40] B. Medjahed, A. Bouguettaya, A. K. Elmagarmid, Composing web services on the semantic web, The VLDB journal 12 (2003) 333–351.

[41] J. Yang, M. P. Papazoglou, Service components for managing the life-cycle of service compositions, Information Systems 29 (2004) 97–125.

[42] J. Cardoso, A. Sheth, Introduction to semantic web services and web process composition, in: International Workshop on Semantic Web Services and Web Process Composition, Springer, 2004, pp. 1–13.

[43] D. Wu, B. Parsia, E. Sirin, J. Hendler, D. Nau, Automating daml-s web services composition using shop2, in: International semantic web conference, Springer, 2003, pp. 195–210.

[44] M. Pistore, A. Marconi, P. Bertoli, P. Traverso, Automated composition of web services by planning at the knowledge level, in: IJCAI, volume 19, 2005, pp. 1252–1259.

[45] S. Mcllraith, T. C. Son, Adapting golog for composition of semantic web services, in: Proc.

International Conference on Knowledge Representation and Reasoning KRR, Toulouse, France, 2002.

[46] R. Hull, Artifact-centric business process models: Brief survey of research results and challenges, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2008, pp. 1152–1163.

[47] R. Sahal, S. H. Alsamhi, K. N. Brown, Digital Twins Collaboration in Industrial Manufacturing, Springer International Publishing, Cham, 2022, pp. 59–72. doi:10.1007/978-3-031-11401-4_7.

[48] J. Wilhelm, C. Petzoldt, T. Beinke, M. Freitag, Review of digital twin-based interaction in smart manufacturing: Enabling cyber-physical systems for human-machine interaction, International Journal of Computer Integrated Manufacturing 34 (2021) 1031–1048. doi:10.1080/0951192X.2021.1963482.

[49] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, M. Mecella, Automatic composition of transition-based semantic web services with messaging, in: VLDB, volume 5, 2005, pp. 613–624.

[50] F. Leotta, M. Mecella, Realizing smart manufacturing architectures through digital twin frameworks (2020).

[51] N. Szabo, Formalizing and securing relationships on public networks, First Monday 2 (1997).

[52] G. Adamson, L. Wang, M. Holm, P. Moore, Cloud manufacturing – a critical review of recent development and future trends, International Journal of Computer Integrated Manufacturing 30 (2017) 347–380. doi:10.1080/0951192X.2015.1031704.

[53] J. E. Kasten, Engineering and manufacturing on the blockchain: A systematic review, IEEE Engineering Management Review 48 (2020) 31–47. doi:10.1109/EMR.2020.2964224.

[54] A. Mavridou, A. Laszka, Designing secure ethereum smart contracts: A finite state machine based approach, in: Financial Cryptography, volume 10957 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 523–540.

[55] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, M. Mecella, Automatic composition of transition-based semantic web services with messaging, in: VLDB, ACM, 2005, pp. 613–624.

[56] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, P. Traverso, Planning and monitoring web service composition, in: Artificial Intelligence: Methodology, Systems, and Applications: 11th International Conference, AIMSA 2004, Varna, Bulgaria, September 2-4, 2004. Proceedings 11, Springer, 2004, pp. 106–115.

[57] R. I. Brafman, G. De Giacomo, M. Mecella, S. Sardina, Service composition in stochastic settings, in: Conference of the Italian Association for Artificial Intelligence, Springer, 2017, pp. 159–171.

[58] G. Edelman, The father of web3 wants you to trust less, wired. com (2021).