Business Process Monitoring on Blockchains: Potentials and Challenges

Claudio Di Ciccio[1], Giovanni Meroni[2](✉), and Pierluigi Plebani[2]

[1] Sapienza University of Rome, Rome, Italy
diciccio@di.uniroma1.it
[2] Politecnico di Milano, Milan, Italy
{giovanni.meroni,pierluigi.plebani}@polimi.it

Abstract. The ability to enable a tamper-proof distribution of immutable data has boosted the studies around the adoption of blockchains also in Business Process Management. In this direction, current research work primarily focuses on blockchain-based business process design, or on execution engines able to enact processes through smart contracts. Although very relevant, less studies have been devoted so far on how the adoption of blockchains can be beneficial to business process monitoring. This work goes into this direction by providing an insightful analysis to understand the benefits as well as the hurdles of blockchain-enabled business process monitoring. In particular, this work considers the adoption of programmable blockchain platforms to manage the generation, distribution, and analysis of business process monitoring data.

Keywords: Blockchain · Business process monitoring · Business Process Management

1 Introduction

Blockchains are gaining momentum in Business Process Management (BPM) research as the infrastructural platform of choice on which collaborative, multi-party business processes are conducted [12]. Thanks to their guarantee of persistence and immutability of the recorded transactions, not only can they operate as a solid backbone for the storage of data and actions, but they are also promising aids for the monitoring of processes that run atop [16].

Research towards the adoption of blockchains for the monitoring of processes is, however, still at its early stages. Thus far, most of the attempts have focused on the generation of readily usable data for the application of existing process mining techniques [5,8,14] and the creation of networks highlighting the most common patterns of exchange of information and assets among peers [6,7,16]. A comprehensive analysis of the aspects of blockchains that may favour or encumber the monitoring of processes is, to the best of our knowledge, still missing.

S. Nurcan et al. (Eds.): BPMDS 2020/EMMSAD 2020, LNBIP 387, pp. 36–51, 2020.
https://doi.org/10.1007/978-3-030-49418-6_3

The goal of this paper is to clarify to what extent a blockchain can be beneficial for business process monitoring. On this basis, the paper identifies a set of research challenges that are worth to be addressed by the research community for the design and realization of blockchain-based process monitoring platforms.

The remainder of this paper is as follows. Section 2 describes the fundamental elements of process monitoring. Section 3 describes the concepts on which blockchain platforms are based and illustrates the main research conducted so far for the process-oriented analysis of blockchain data. Section 4 examines the challenges and opportunities we envision for a blockchain-based process monitoring architecture. Finally, Sect. 5 concludes the paper.

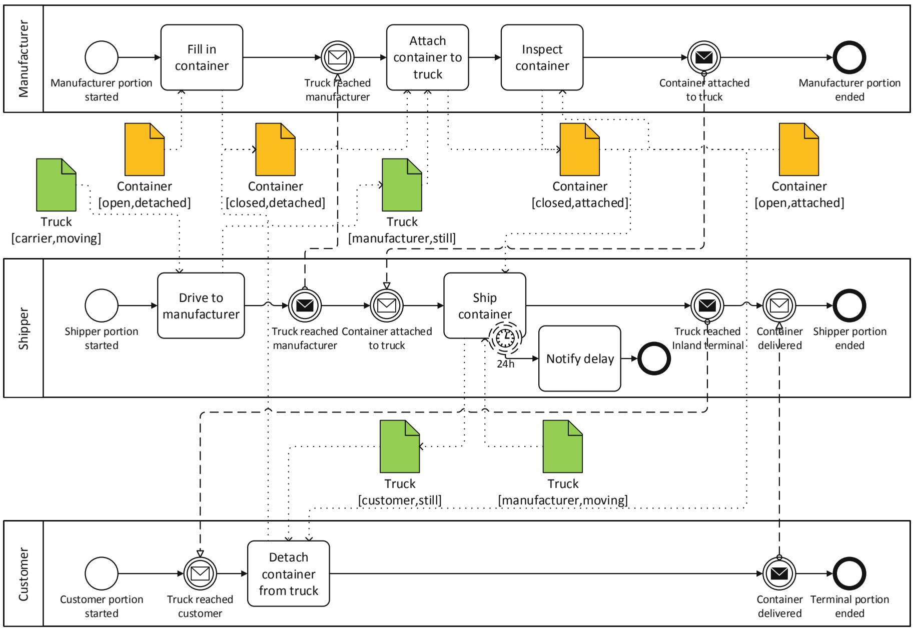
Manufacturer
Manufacturer portion started
Fill in container
Truck reached manufacturer
Attach container to truck
Inspect container
Container attached to truck
Manufacturer portion ended
Container [open,detached]
Container [closed,detached]
Container [closed,attached]
Container [open,attached]
Truck [carrier,moving]
Truck [manufacturer,still]
Shipper
Shipper portion started
Drive to manufacturer
Truck reached manufacturer
Container attached to truck
Ship container
24h
Notify delay
Truck reached Inland terminal
Container delivered
Shipper portion ended
Truck [customer,still]
Truck [manufacturer,moving]
Customer
Customer portion started
Truck reached customer
Detach container from truck
Container delivered
Terminal portion ended

Fig. 1. Motivating example.

2 Business Process Monitoring

Business process monitoring aims at identifying how well running processes are performing with respect to performance measures and objectives. Depending on the available tools and data, a business process platform can report on the running processes, from the sole tracking of the running instances to the checking of deviations with respect to the expected behaviour and the identification of other anomalies. This section briefly introduces the main characteristics of business process monitoring platforms in terms of the possible objectives of the monitoring (i.e., the why), the available techniques (i.e., the how), and the subject of monitoring (i.e., the what).

To better explain these aspects, we use an example taken from the logistics domain. Figure 1 shows the Business Process Model and Notation (BPMN) model of a shipment process in particular. The example involves a manufacturer, M, who receives an order from one of its customers, C, and a shipper, S, on whom M relies for the delivery of the goods to C. At first, M starts filling a shipping container with the goods requested by C. Meanwhile, S starts driving one of its trucks to M's production facility. Once the truck arrives, M firstly attaches the container to the truck, then inspects the container to verify if all the goods requested by C are present. Such an inspection should be performed only at this stage, and the container should not be opened again until it reaches the premises of C. Once the inspection completes, S ships the container to C, which detaches it from the truck. In case the shipping activity takes longer that 24 h, S must justify the delay.

Why to Monitor. There are several reasons why a monitoring platform should be introduced. As a general need, the process owner and the recipients are interested in verifying and demonstrating that the process is behaving correctly. A monitoring platform can be a passive element that merely records the performed actions, or it can actively contribute to handle the occurring deviations.

Moreover, the objectives of a business process monitoring platform can be various: to determine if activities take longer than expected to complete, if there are bottlenecks in the process, if resources are under- or over-utilized, and if there are violations in the process execution, among other things. Depending on the needs of the process owner, all – or a subset – of these aspects can be considered.

How to Monitor. According to the classification proposed by [1] and [17], process monitoring techniques can be classified in five main groups: event data logging, Business Activity Monitoring, runtime performance analysis, conformance checking, and compliance checking.

Event data logging is the generation of sequences of events related to a specific process instance being executed. Events can provide notifications on the activities being executed, or on the artifacts (i.e., the physical or virtual objects manipulated by the process) and the resources (i.e., the human operators or software components responsible for executing activities) participating in the process. Once collected, events are typically stored in so-called event logs. Since several other monitoring techniques require event data to work, this technique is often seen as a prerequisite for them.

Business Activity Monitoring (BAM) and Runtime Performance Analysis: Also known as "monitoring" [1], BAM analyzes real-time information on the activities being executed (e.g., response time and failure rate). With this technique it is possible to measure Key Performance Indicators (KPIs) relevant for the process, thus determining how well activities are performed. Given event data, BAM produces measurements for KPIs. Instead, runtime performance analysis focuses on the data analysis of performance information on the processes being executed to identify bottlenecks or resource allocation problems. Unlike BAM, which focuses on single activities, Runtime performance analysis focuses on process runs, thus accounting for dependencies among activities. Given a process model and event data, runtime performance analysis produces performance-related diagnostic information.

By resorting on BAM, and runtime performance analysis, it is possible to measure KPIs and identify other issues not directly related to the process structure. For instance, it is possible to determine if activity *Attach container to truck* is causing a bottleneck, or if *Drive to manufacturer* consumes too many resources (e.g., if M is located in a poorly connected area). Based on the agreements between organizations, these techniques may be confined only to activities belonging to each organization, or they may be applied to all the activities.

Conformance checking consists in the techniques that compare the modeled process behavior with the one evidenced by execution data. To this end, the gathered event data are replayed on the process model, so as to detect deviations from the expected behaviour. Given a process model and event data, Conformance checking produces conformance-related diagnostic information.

With conformance checking, the stakeholders can verify if the execution is in line with the process description. In particular, the nature of the model plays an important role in defining the degrees of freedom that are left to the process executors. A collaboration diagram (e.g., the complete collaboration diagram in Fig. 1) will force the whole process to strictly adhere to the specifications. A process diagram (e.g., only the portion of the process inside a specific pool) will force the process portion belonging to that stakeholder to adhere to the specifications. Finally, a choreography diagram will force only the interactions among stakeholders to adhere to the specifications, leaving the stakeholders free to alter their internal processes.

Compliance checking encompasses the techniques aimed at verifying that constraints representing regulations, guidelines, policies and laws, are fulfilled by the process. It differs from conformance checking because constraints focus on process rules, rather than on entire process runs.

Through compliance checking techniques, it is possible to define complex constraints on the process that predicate both on the structure and on non functional aspects. Instead of relying on a process model, compliance checking relies on compliance rules that describe only the elements of the process that are useful to assess the constraint. For instance, it is possible to monitor if the container is delivered to C within two days since when M finished preparing it. Likewise, it is possible to monitor if less than 1% of the shipments were carried out without inspecting the container. To this aim, according to [10], several compliance checking techniques and languages exist. Since constraints predicate on specific portions of the process, rather than on the process as a whole, it is much easier for stakeholders to agree on monitoring them. In fact, only activities required for the assessment of such constraints have to be disclosed, thus overcoming one of the issues of conformance checking.

What to Monitor. Depending on the monitoring technique and on the underlying representation of the process to monitor, different kinds of events have to be logged for the monitoring to be reliable. Conformance checking techniques typically require events notifying the start or termination of activities, or the transmission and receipt of messages among participants. BAM, runtime performance analysis and compliance checking techniques usually require more complex events, also indicating when artifacts were manipulated or who performed a task (e.g., starting an activity or modifying an artifact).

Typically, if the reference model adopted for process monitoring is a *process diagram*, only events belonging to the owner of the process are collected and analyzed. When the process consists only of either automated activities or form-based ones, obtaining events is a relatively easy task. In fact, event logs can be retrieved from the Business Process Management System (BPMS) in charge of executing the process. Also, since users are required to interact with the BPMS to perform business activities, event logs contain accurate information on who performed which task, when the task was performed, and which artifacts were involved during its execution. However, when the process also involves manual activities, that is, activities that are performed by users without interacting with the BPMS (e.g., shipping the container), collecting reliable event logs becomes challenging. In fact, users may forget to notify to the BPMS when they perform activities, they may incorrectly indicate in the notifications when the activities were performed, they may indicate that they performed an activity which was not done or which was done by another user. These issues can be partially solved with Internet of Things (IoT)-based solutions, such as artifact-driven monitoring [13] or Unicorn [2], which autonomously collect events from the artifacts being manipulated, to be then analyzed to infer which activity was executed.

If the process to monitor is represented as a *collaboration diagram*, events belonging to all the involved organizations have to be logged and shared among participants. This is a challenging task both from the organizational and technical standpoint. However, from an organizational viewpoint, the participants may be reluctant to share events on the activities being performed, as it may allow competitors to uncover their operations. In case of BAM and runtime performance analysis, sharing such events may even violate privacy regulations, such as the General Data Protection Regulation (GDPR), since information on the employees performing activities may be shared to the other organizations. Technical-wise, sharing events typically requires either individual information systems to be federated, or a centralized cross-organizational information system to be deployed and adopted by all the participants. To partially overcome these issues, organizations can autonomously monitor their own portions, and then share aggregated monitoring data to the other participants. However, this approach reintroduces the problem of trust between organizations, moving it from the execution to the monitoring of the processes. In fact, for this approach to hold, organizations are required to trust each other, assuming that monitoring data reflect the actual behavior of the process.

In case the process is represented as a *choreography diagram*, events related to the transmission and reception of messages between organizations have to be logged. From the technical standpoint, as long as the message exchanges are performed digitally (e.g., email, web service invocations), it is relatively easy to log and distribute events. In fact, it is sufficient to passively monitor the communication channels, generating events whenever communication activity is detected. On the other hand, if physical objects are exchanged, generating event logs is a more complex task. In fact, an active agent is required to observe the real world and produce an event whenever some physical object is either received or sent. Originally, this was done by relying on human operators, but it suffered from the same limitations as the ones outlined for manual activities. Therefore, IoT-based solutions to track physical objects are adopted as long as the contents of the messages are kept confidential, and only events relevant for the process being monitored are disclosed.

Finally, in case compliance rules have to be monitored, depending on the language and technique adopted, events related to messages, activities, or artifacts have to be logged. Consequently, compliance checking has the same technical limitations as all the conformance checking techniques. However, not every event related to the process has to be logged, but only the ones required for verifying the compliance rules. Therefore, monitoring has a much lower footprint on the organizations. Also, since organizations can selectively choose which events to be logged and made available to the other participants, they can agree on not to share information that discloses their know-how.

3 Monitoring with the Blockchain: State of the Art

In this section, we summarize the fundamental notions on which blockchain platforms are based and the research conducted thus far that aims at analysing blockchain data for process execution and analysis.

3.1 Elements of Blockchains

A blockchain is a protocol for the decentralized storage of a tamper-proof sequence of transactions, maintained and verified by the nodes participating in the network. A *ledger* is an append-only list of data units named *transactions*. Every transaction records a transfer of value (digital assets, cryptocurrencies, information bits, etc.) between two accounts. The sender cryptographically signs the transaction to provide evidence that it is not counterfeit. Blockchains such as Bitcoin [15] and Ethereum [21] collate transactions into so-called blocks. Blocks are thus used as the messages to be broadcast to every node. The order among blocks (and, a fortiori, the transactions therein) is kept by a hash-based backward linking: every block keeps the digest of a hashing function applied to the previous block. All together, the links generate a chain-like structure: hence the name blockchain. Locally to a node, transactions are subject to a total ordering relation: the evolution of the state of the parties' accounts depend on the sequence of operations recorded in the ledger. Blocks are, in fact, a measure of time as their addition to the chain determines the passage to the next global system state. To pay back the effort of nodes, an economic incentive is proposed that distributes so-called cryptocurrencies to the nodes that publish the accepted blocks. Nodes participating in the network guarantee that transactions and blocks are valid and thus prevent the data structure to be tampered with. Also, the replication of the ledger makes it possible to have the stored information always available locally to every node. However, the ledger may differ from node to node: the nodes reach *eventual consensus* on the correct sequence in the ledger. Temporary divergences between the local images of the ledger are called *forks*. The way in which access and right to write are granted, determine two main categorisations of the blockchain platform in use: *private* blockchains are accessible only to a restricted number of peers, as opposed to the *public* ones; if a selected number of participants only is allowed to decide on the next blocks, the blockchain is *permissioned*, otherwise it is *permissionless*. Natively, Bitcoin and Ethereum are natively public permissionless blockchain, although for the latter private networks can be created that operate within conortia, allowing only a subset of nodes to mine blocks. Hyperledger Fabric,[1] instead, is conceived as a consortium (private) permissioned blockchain.

Second-generation blockchains such as Ethereum and Hyperledger Fabric support the so-called *smart contracts* [18], that is, executable code expressing how business is to be conducted among contracting parties (e.g., transfer digital assets after a condition is fulfilled). In this paper, we will focus on this kind of blockchains operating as distributed programmable platforms. Smart contracts often require data from the world outside the blockchain sphere (e.g., financial data, weather-related information, random numbers, sensors from hardware devices). However, they cannot directly invoke external APIs. Therefore, smart contracts need software adaptors that play that interfacing role. Those artefacts are named *oracles* [22]. Oracles can be further classified as *software* or *hardware* oracles. Software oracles aim to extract information from programmed applications (e.g., web services), whereas hardware oracles extract data from the physical world (e.g., IoT devices).

3.2 Current Approaches

To date, preliminary attempts have been proposed that can be the basis to be built upon for process monitoring in the blockchain. Smart contracts allow for the codification of business process logic on the blockchain, as shown in the seminal work of Weber et al. [20]. Later, a similar approach has been applied within the Caterpillar [9] and Lorikeet [19] tools, as well as by Madsen et al.

[1] https://www.hyperledger.org/projects/fabric.

[11]. As several modern Business Process Management Systems (BPMSs) do, those approaches adopt a Model-Driven Engineering (MDE) paradigm to let the process analysts provide graphical representations of the process and turn it into executable code enacting it [4].

From the monitoring perspective, the efforts have been mostly devoted to event data logging thus far: the main rationale is to extract and process the payload of transactions to turn them into event logs that are readily available for process mining tools. The ordering of events is based upon the ordering of the transactions in the ledger, whereas the attributes of the event (activity name, timestamp, resource, and the like) are identified based on the signature of the invoked function on the smart contract [14], a user-defined descriptor (manifest) [8], or the change of the smart contract's attribute value [5]. Thereupon, process mining techniques (including conformance checking) are held to analyse the generated event logs.

Other approaches have been applied to analyse blockchain-mediated communications among peers, such as GraphSense [7]. Filtz et al. [6] studied the graph of addresses in Bitcoin and thereby examined the transaction behavior of users, taking into consideration exchange rates between virtual and fiat currencies. Prybila et al. [16] focus in particular on the transposition of handovers of tasks in a process to Bitcoin transactions. With their software prototype, they are thus able to verify the execution flow of a process by tracking the transactions exchanged among peers.

The research conducted thus far constitutes a clear advancement towards future architectures for business processes monitoring that are based on the blockchain. In the following section, we discuss challenges and opportunities that come along with their design and adoption.

4 Monitoring with Blockchain: Challenges and Opportunities

Due to its properties, a blockchain can be adopted as a distributed infrastructure on top of which a new type of business process monitoring platforms can be built. These platforms can exploit the properties of data immutability, trust among the parties, and data distribution offered by design by blockchains. Owing to their programmability, we focus in particular on second-generation blockchains in the remainder of this paper.

To properly describe the potentials of introducing blockchains to this end, Fig. 2 shows a reference architecture that couples the typical BPMS monitoring subsystem with a blockchain. Without focusing on one of the specific monitoring approaches discussed in Sect. 2, we can generalize the input of the monitoring platform with the status of the process instances and the ongoing activities (*process status)*, and of the artifacts managed thereby (*artifact status*). Depending on the adopted technique, the logic implemented by the monitoring platform transforms this information about the status. The produced output, generally referred as *monitoring data*, can be a collection of event logs, or a transformation

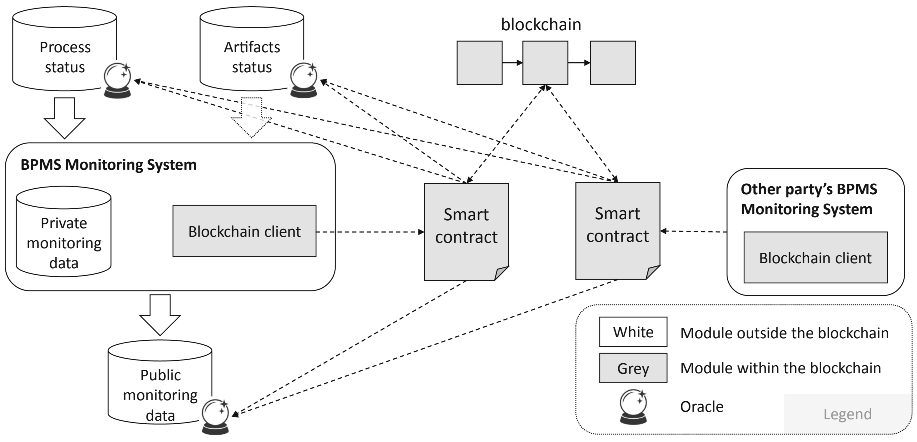

Fig. 2. Blockchain-enhanced BPMS monitoring reference architecture

of the obtained input data into aggregated information that is more meaningful to the analysis (e.g., at a higher level of abstraction than low-level events). A portion of this output can be kept private (*private monitoring data*) or made public (*public monitoring data*), to let other interested parties to check, for instance, the compliance of the process.

When enriching the monitoring platform with a blockchain, the monitoring logic may be encoded in one or more *smart contracts*. First of all, this requires that the monitoring system includes a *blockchain client* which enables the communication with the rest of the blockchain infrastructure. Secondly, as the output of a smart contract is published as a transaction payload on the blockchain, the resulting monitoring data produced by the smart contract is automatically available to anyone allowed to access the blockchain. This implies that the monitoring logic implemented as smart contract must be limited to the part producing public monitoring data. On the one hand, this opportunity increases the transparency of the monitoring and the possibility for external actors to evaluate the behaviour of the process, as the smart contract is immutable and executed on all the nodes in the blockchain network. On the other hand, since the publication of data on the blockchain has an impact in terms of cost and performance, it becomes of utmost importance to establish which monitoring data can be included in the blockchain (i.e., on-chain thus trusted by definition), and which one can be left off-chain as typical public monitoring data (i.e., only under the control of the party producing them).

The distinction between data on- and off-chain is relevant not only when considering the output of the monitoring platform, but also concerning the input of a smart contract as they can natively operate only on data published on the blockchain. To overcome this limitation, blockchains offer *oracles* to extend the smart contract accessibility to off-chain data. For this reason, it is required that any dataset to which a smart contract requires access for its computation, needs to be coupled with an oracle. Since coupling a dataset with an oracle means making those data visible to all the members of the blockchain, this implies that the data set should be properly partitioned into public (i.e., accessible through the oracle) and private parts.

Table 1. Challenges in monitoring with the blockchain

Aspect	Challenge	Perspective		
		Design rationale	Lack of capability	Trust management
Smart contracts	Monitoring transparency	X		X
	Observability	X		
	Lack of reactivity	X	X	
Oracles	Time management		X	X
	Reliability	X		X
	Flexibility	X	X	
Data management	Data quality		X	X
	Data size	X		
	Side effects		X	

In light of the above discussion, we observe that, to improve the business process monitoring via blockchain, a thorough blueprint is required in terms of the smart contracts (reflecting the monitoring capabilities), the oracles (identifying the data sources), and the monitoring data (balancing between on-chain and off-chain data). Table 1 illustrates the research challenges related to those three aspects and categorizes them according to three main perspectives: the need for a thorough *design rationale* behind the realisation of the monitoring infrastructure; the necessity to tackle the *lack of capabilities* that the infrastructural usage of the blockchain brings; the demand for a policy of *trust management* with information and actors. Each of the following subsections discusses in further detail those aspects and the related challenges, emphasizing the perspectives from which the issues are analyzed.

4.1 Challenges About Smart Contracts

Monitoring Transparency. To improve the transparency of compliance checking, especially in case of multi-party business processes, a smart contract holds a crucial role. In fact, based on the information that can be made accessible through oracles and the relevant transactions mined in the blockchain, a smart contract can analyse the current status of the process enactment and verify if the control flow – in the case of orchestrated processes – or the message exchange – in the case of a choreographed process – are behaving as expected. As the code composing the smart contract is immutably stored on the blockchain, and it is executed on all the blockchain clients to reach consensus, it is extremely hard for a single party to alter it in order to counterfeit the result. As a consequence, when a single party is involved in the business process, being the smart contract published on the blockchain, the logic that drives compliance becomes publicly available to all the parties interested in the soundness of the process, even if they are not directly involved in the enactment (e.g., auditors). Conversely, in multi-party business processes, information about the obligations involving the parties can be produced and observed with smart contracts. For example, by turning the process model in Fig. 1 into a smart contract, M, S and C agree on how the process should be carried out. As the constraints become public, none of them can complain that they expected the process to be executed differently.

If, on the one hand, such a transparency offered by smart contracts increases the trust in the process execution, on the other hand, it requires smart contracts to be properly *designed* in order to expose only the information that should be made available to external actors. Moreover, the ability for a smart contract to verify possible deviations in the process enactment is strictly related to the monitoring data that are available through the oracles. As a consequence, the availability of proper data sources that can be accessed by the smart contract is fundamental. With single-party business processes, this issue is not so critical as the party is responsible for designing the smart contract as well as for designing the oracles and choosing the data sources. Conversely, when talking about multi-party business processes, an agreement among parties must include also the possibility to make available some of the data about the process and artifact status to other parties. This opens an additional issue about the accessibility of those data. For instance, to determine if the process portion carried out by S is correctly performed, S must expose the information on the position and speed of its trucks.

In a blockchain, the transactions of mined blocks are available to all participants. Consequently, additional mechanisms must be implemented on top of the blockchain (e.g., based on encryption) to limit the visibility of this data only to the subset of clients that are actually allowed to see them, so as to enable *trust*. For example, S may agree on sharing information on its trucks with M, but may refuse to make this information publicly available, as competitors may exploit it (e.g., by finding areas that are not well covered by S).

Observability. Although both the smart contracts and the invocations of its methods are stored in the blockchain, and their execution can be performed and analyzed by any participant, most blockchains require smart contracts to explicitly define methods to retrieve their information. In other words, variables that are used by smart contracts are accessible only by the smart contract itself, unless methods to make their contents available are explicitly defined in the smart contracts *design*. As a consequence, before putting in place a blockchain-based monitoring platform, care should be taken defining which information can be retrieved from the smart contract. For example, suppose that, to monitor the process in Fig. 1, a smart contract is implemented that has an internal representation of the process and of the status of each activity. That smart contract may expose a function to check whether the process conforms to the model or not, without providing information on the activities. As a consequence, although the smart contract internally knows that, e.g., *Ship container* is running and *Attach container to truck* is complete, it would lack a way to communicate this information to other smart contracts or other participants, which cannot rely on it to determine the status of the process and its activities.

To mitigate this issue, one could "debug" a smart contract by tracing the execution of each transaction since when the deployment took place, thus identifying the variables and how they change over time, similarly to the approach of Duchmann and Koschmider [5]. However, if the discovered information is required by another smart contract, this information should be provided off-chain even though it originated on-chain, with consequent trust issues and the need, once again, to rely on an oracle.

Lack of Reactivity. A smart contract *lacks the capability* of independently making calls or invocations to endpoints outside the blockchain upon the verification of certain conditions. This is a limitation for business process monitoring, as in case of deviations, the process owner or the parties involved in the process wish they were informed in order to properly react. For example, if the smart contract monitoring the process in Fig. 1 detects that activity *Inspect container* was performed while the container was being shipped, it cannot autonomously contact S off-chain to request a justification for that action.

To solve this issue, a smart contract has to be *designed* so as to either expose public methods that can be periodically called by the interested parties to check if some deviations occurred, or emit events and require parties to constantly monitor the blockchain in order to catch them as soon as they fire. In both cases, e.g., it is S's duty to constantly check for events notifying an anomaly in its own process and promptly react to them.

4.2 Challenges About Oracles

Time Management. Among the several aspects that are interesting to monitor about a business process, one of the most pivotal is checking if an activity, or a group of them, is performed on time. Nevertheless, implementing a smart contract able to verify this condition could be cumbersome as a blockchain *lacks a notion* of time aside from the coarse-grained block time [14]. More in detail, although a blockchain sorts the transactions, it cannot deal with timers. This is due to the fact that the expiration of a timer, or more simply a clock-ticking event, would be an action that originates from the smart contract itself. However, as a smart contract can only perform actions that are externally invoked, such actions cannot be performed without the help of an external entity. For example, suppose that a smart contract is adopted to check whether activity *Ship container* is executed on time. That smart contract cannot determine that activity *Ship container* took longer than 24 h until it receives a notification that the activity was completed, unless it is actively polled by an external entity.

For this reason, time must be managed externally to the blockchain by means of specific oracles which must be configured by the smart contract to send a trigger whenever a timeout expires. It is also important to consider that those oracles are external to the blockchain by definition, hence outside the chain of *trust* managed by the blockchain. For this reason, when designing a time oracle, the situation in which the oracle experiences a failure or produces fake data (e.g., it goes out of sync) must be taken into consideration. To mitigate this issue, oracles may integrate time synchronization protocols.

Reliability. The goal of an oracle is to allow the smart contracts to acquire information from the real world. Thus, oracles must guarantee the correctness of the data they emit. However, this may not occur for two reasons. Firstly, the oracle may deliver data that are intentionally wrong or – because of a man-in-the-middle attack – data are forged before being sent to the smart contract. Secondly, the oracle may not be reliable and the data produced could be accidentally wrong. For example, if the truck's GPS receiver is breached, the related oracle could send incorrect information on its location.

Both circumstances hamper the *trust* in the gathered data. The solution is to rely not on a single oracle to obtain information about a phenomenon occurring in the real world, but to have a set of oracles, possibly managed by different actors. With such a *design*, the effort to cheat on the smart contract becomes significant as it requires to forge several oracles. Moreover, the smart contract can query several oracles and – assuming that problems may occur only on a minority of them – compare the data being sent to determine which ones cannot be trusted. For instance, the smart contract may rely on information coming from the truck's GPS receiver, the truck driver's smartphone, and the highway tollbooths, to know the location of a truck. Although this approach could solve the problem, having a set of oracles for the same phenomena is not always feasible or affordable, especially when monitoring human-based activities.

A possible solution to the problem of trust is to certify the oracles. In this sense, approaches similar to the Public Key Infrastructure (PKI) can be adopted to introduce authentication and authorization mechanisms.

Flexibility. Adopting oracles to allow smart contracts to check the behaviour of a process implies that all the phenomena relevant for the monitoring should be exposed through oracles. Since the smart contract should know in advance which are the oracles providing the needed data, this could result in a *lack of flexibility*. In fact, adding new oracles after the monitoring has been designed could be useless, as there is no possibility to inform the smart contract about their existence. For example, suppose that the monitoring platform relied initially on manual notifications to determine when the container was filled in, and references to that oracle were hardcoded in the platform's smart contracts. If later on containers are equipped with scales to automatically infer if they are full or empty, it is not possible for the platform to rely on that information, unless smart contracts are redesigned and deployed anew.

Mechanisms for enabling late binding of oracles to smart contracts are thus desirable for a proper *design*. Notice that late binding would also tackle problems of reliability. Without that mechanism in place, an oracle that is no longer available cannot be replaced.

4.3 Challenges About Monitoring Data Management

Data Quality. In addition to the problems discussed in the previous section related to the possibility that the oracles are not able to provide reliable data, there is also the possibility that the data provider used by the oracle itself is *not trustworthy*. For instance, in the case of a manual activity, it might happen that the oracle is not connected to any sensor, as it is not possible to automatically get the information, but the change in the status of the activity is personally done by the operator. Consequently, the operator could cheat the system declaring, for instance, that an activity is concluded even though it is not the case. Although this is a well-known problem in business process monitoring, we are confident that also the adoption of blockchains may not be beneficial to solve it.

Furthermore, if erroneous data are stored in the blockchain, they can be amended only by appending the correct information, as the blockchain *does not allow* for the alteration of data in a mined block. Therefore, effective mechanisms to assess the quality of monitoring data during the consensus phase are key [3].

Data Size. In a blockchain, the larger the amount of stored data is, the more expensive the transaction gets. This simple rule has a significant impact on monitoring costs. Indeed, in the initial approaches [8,14], all the data that could be useful for monitoring were supposed to be stored on-chain. Nevertheless, to reduce these costs, care should be taken in the *design* of the smart contract to minimize the amount of on-chain information to the sole data that are required to perform monitoring [8]. To this aim, distributed file systems such as IPFS[2] can be adopted to store the entire monitoring data set. Then, the transaction only includes a link to externally stored data, and a hash value computed to guarantee immutability. However, as smart contracts cannot natively retrieve and process off-chain data, this could imply that oracle-mediated operations are required again.

Side Effects. Most blockchains are prone to soft forks, i.e., branches in the chain of blocks caused by two or more blocks pointing to the same predecessor. To solve ambiguities, blockchain clients consider as valid the longest chain, that is, the one having the highest number of subsequent blocks originating from the point of forking. From a monitoring standpoint, this *lack of information consistency* is an issue, since valid monitoring data may not be considered as the block containing them happen to lie on a discarded post-fork branch.

Aside from soft forks, public blockchains such as Ethereum are also prone to so-called hard forks. In case a change in the consensus protocol is made

[2] Interplanetary File System (IPFS), https://ipfs.io.

– for either technical or political reasons – some participants may not accept it. Unlike soft forks, hard forks cause a split in the blockchain network, which hampers interoperability. From the monitoring standpoint, hard forks may break the platform if some participants decide not to migrate to the new protocol.

5 Conclusion

Throughout this paper, we have discussed the advantages and challenges that come along the interplay between blockchain data and process analysis for monitoring. Despite the growing interest in the adoption of blockchain technologies for process execution environments, research in that direction is still at its early stages. Considering a reference architecture for the realisation of blockchain-based process monitoring, we have focused on the role that smart contracts, oracles and data management strategies play, in pursuit of a fruitful discussion in the community that drives the adoption of blockchain in process monitoring.

Acknowledgements. The work of Claudio Di Ciccio was partly supported by the MIUR under grant "Dipartimenti di eccellenza 2018–2022" of the Department of Computer Science at Sapienza University of Rome.

References

1. van der Aalst, W.M.P.: Business process management: a comprehensive survey. ISRN Softw. Eng. **2013**(507984), 37 (2013)
2. Beyer, J., Kuhn, P., Hewelt, M., Mandal, S., Weske, M.: Unicorn meets Chimera: integrating external events into case management. In: Proceedings of the BPM Demo Track, pp. 67–72 (2016)
3. Cappiello, C., Comuzzi, M., Daniel, F., Meroni, G.: Data quality control in blockchain applications. In: BPM (Blockchain and CEE Forum), pp. 166–181 (2019)
4. Di Ciccio, C., et al.: Blockchain support for collaborative business processes. Informatik Spektrum **42**, 182–190 (2019)
5. Duchmann, F., Koschmider, A.: Validation of smart contracts using process mining. In: ZEUS, pp. 13–16 (2019)
6. Filtz, E., Polleres, A., Karl, R., Haslhofer, B.: Evolution of the bitcoin address graph. In: Haber, P., Lampoltshammer, T., Mayr, M. (eds.) Data Science - Analytics and Applications, pp. 77–82. Springer, Wiesbaden (2017). https://doi.org/10.1007/978-3-658-19287-7_11
7. Haslhofer, B., Karl, R., Filtz, E.: O bitcoin where art thou? Insight into large-scale transaction graphs. In: SEMANTiCS (Posters, Demos) (2016)
8. Klinkmüller, C., Ponomarev, A., Tran, A.B., Weber, I., van der Aalst, W.: Mining blockchain processes: extracting process mining data from blockchain applications. In: BPM (Blockchain and CEE Forum), pp. 71–86 (2019)
9. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., Ponomarev, A.: Caterpillar: a business process execution engine on the Ethereum blockchain. Sofw. Pract. Exp. **49**(7), 1162–1193 (2019)

10. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: functionalities, application, and tool-support. Inf. Syst. **54**, 209–234 (2015)
11. Madsen, M.F., Gaub, M., Høgnason, T., Kirkbro, M.E., Slaats, T., Debois, S.: Collaboration among adversaries: distributed workflow execution on a blockchain. In: FAB, pp. 8–15 (2018)
12. Mendling, J., et al.: Blockchains for business process management - challenges and opportunities. ACM Trans. Manag. Inf. Syst. **9**(1), 4:1–4:16 (2018)
13. Meroni, G., Baresi, L., Montali, M., Plebani, P.: Multi-party business process compliance monitoring through IoT-enabled artifacts. Inf. Syst. **73**, 61–78 (2018)
14. Mühlberger, R., Bachhofner, S., Di Ciccio, C., García-Bañuelos, L., López-Pintado, O.: Extracting event logs for process mining from data stored on the blockchain. In: Di Francescomarino, C., Dijkman, R., Zdun, U. (eds.) BPM 2019. LNBIP, vol. 362, pp. 690–703. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37453-2_55
15. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008). https://bitcoin.org/bitcoin.pdf
16. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime verification for business processes utilizing the bitcoin blockchain. In: FGCS (2017)
17. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30409-5
18. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997). https://firstmonday.org/ojs/index.php/fm/article/view/548
19. Tran, A.B., Lu, Q., Weber, I.: Lorikeet: a model-driven engineering tool for blockchain-based business process execution and asset management. In: BPM Demos, pp. 56–60 (2018)
20. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 329–347. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_19
21. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger (2018). https://ethereum.github.io/yellowpaper/paper.pdf
22. Xu, X., Weber, I., Staples, M.: Architecture for Blockchain Applications. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-03035-3