# Trusted Artifact-driven Process Monitoring of Multi-Party Business Processes with Blockchain

## (Pre-print version)

Giovanni Meroni[1], Pierluigi Plebani[1], and Francesco Vona[1]

Politecnico di Milano – Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza Leonardo da Vinci, 32 - 20133 Milano, Italy
{giovanni.meroni,pierluigi.plebani}@polimi.it,
francesco.vona@mail.polimi.it

**Abstract.** Multi-party business processes are characterized by the lack of a central coordination, as each participant controls only a portion of the process. Nonetheless, organizations often need to know how the whole process is performed, especially when artifacts belonging to an organization are manipulated by the other participants. This requires a monitoring system able to collect and share in a trusted way data about the status of the activities performed by the different parties. To achieve this goal, in this paper we combine artifact-driven monitoring with blockchain. The former, introduced in previous work, can determine how the process is executed, while the latter enables a trusted data exchange among the participants of the business process to reduce the possibility for a fraudulent organization to alter monitoring data. The feasibility and the impacts on costs of the proposed platform is validated via a prototype based on the Ethereum blockchain implementing a real-world use case.

**Keywords:** blockchain, distributed ledger, ethereum, artifact-driven monitoring, trusted process monitoring, cyber-physical systems

## 1 Introduction

Business process monitoring holds a fundamental role in the Business Process Management life-cycle. In fact, monitoring does not only allow checking the compliance of the running process with respect to the expected behaviour, but also collecting data that are useful to improve the process model for future enactments. Especially in case of multi-party business processes, process monitoring becomes very difficult. This is due to the fact that each party has visibility on a portion of the whole process. Therefore, is up to the party involved in the possibly failing activities to notify issues to the other parties. This is particularly crucial when the failing activities operate on resources that belong to another party. For instance, in the logistic domain, the sender of a product wants to be informed about the

way its package is manipulated by the shippers involved in the delivery until it arrives at destination. To cope with this issue, current solutions usually rely on a centralized architecture, assuming that a specific entity is in charge of supervising the entire process execution by collecting all the relevant information on the status of the tasks and on the resources given by the parties [4].

In this scenario, artifact-driven process monitoring is an approach that has been proposed to monitor business processes [9]. It does not require any central authority as the monitoring is performed from the standpoint of the resources, i.e., the artifacts, managed by the parties during the execution of the process. Instead of relying on explicit notifications sent by human operators, artifact-driven process monitoring relies on the conditions of the physical objects (i.e., artifacts) participating in a process to determine when business activities are executed. Together with the Internet of Things (IoT) paradigm, which turns physical objects into smart entities aware of their conditions and of the process they participate in, artifact-driven process monitoring allows to autonomously monitor the process, regardless of the organization or the person in charge of executing it. In addition, artifact-driven process monitoring relies on a declarative representation of the process to monitor. This makes the monitoring more flexible, and able to handle deviations and violations that may occur at run-time without interruptions or human intervention. Moreover, flexibility is also improved because a central authority – which could became a bottleneck – is no longer required.

To this aim, artifact-driven process monitoring solves the issue of knowing the conditions of the physical objects, and the execution of the process. However, it requires the organizations to *trust* each other. In fact, a malicious organization may intentionally alter monitoring information collected by the smart objects, and then it may claim that the process was executed differently than what it actually was. While a previous work investigated the possible connections between artifact-driven monitoring and blockchain [11], this paper investigates the trade-off between the assurance of having persistent monitoring data and the minimization of the data written on the blockchain, which has been validated through a prototype based on Ethereum [17]. Real-world processes – and the related monitoring data – are also used to evaluate issues related to the cost of public blockchains in terms of cryptocurrency.

The remainder of this paper is structured as follows: Section 2 outlines the requirements that a trusted monitoring platform should fulfill. Section 3 presents and compares the architecture of the two possible blockchain-based solutions. Section 4 validates the proposed architectures against a real-world use case. Section 5 surveys related work. Finally, Section 6 draws the conclusions of this work and outlines future research plans.

## 2 Trusted Process Monitoring

Like in every multi-party business process, no organization has full control on the whole process. Instead, each organization is responsible only for the activities assigned to it. Consequently, being able to monitor the whole process becomes

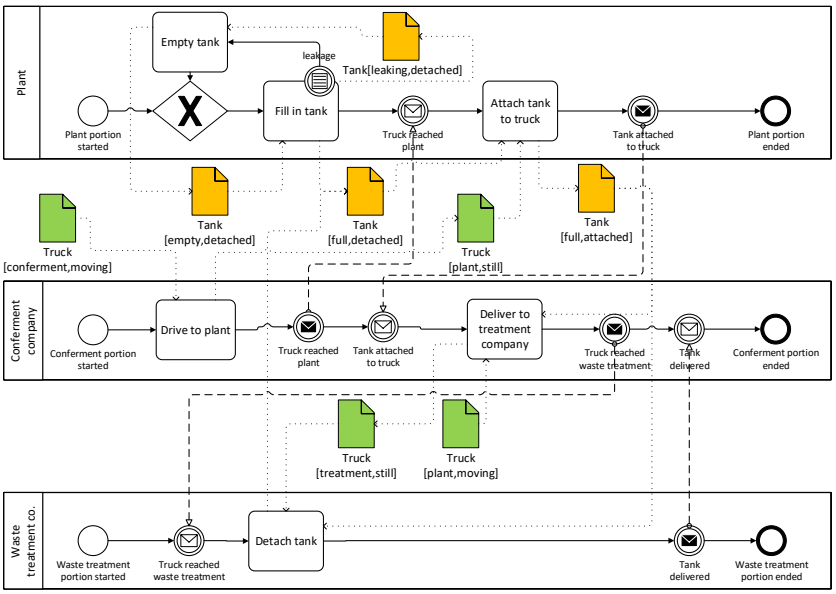


Fig. 1: BPMN diagram of the dangerous goods shipment process.

critical for organizations to be sure that the process is executed as expected and, if not, to identify who is responsible for such an inconsistency.

To better understand the importance of a reliable and trusted process monitoring solution, a case study concerning the shipment of dangerous goods is presented. An industrial plant $P$, to dispose of chemical waste, relies on the conferment company $C$. In turn, $C$ relies on the waste treatment company $T$ to neutralize the waste. The disposal process, which is represented in Figure 1 using Business Process Model and Notation (BPMN), is organized as follows. Firstly, $P$ stores the chemical waste inside a tank and waits for $C$ to reach its plant. If the tank has a leakage, to avoid the chemical to be spread and pollute the environment, $C$ has to immediately empty it and use another tank. Once $C$ arrives, the tank is attached to its truck, then $C$ leaves the plant and delivers the tank to $T$. Finally, $T$ detaches the tank.

Concerning the trust, imagine that the tank is not properly disposed. Instead, it is abandoned in the woods and, after some time, it is found by the forest rangers. Without knowing how the process was actually performed, it would be impossible for them to know who is responsible for this crime. Firstly, $P$ may have completely skipped the conferment part, and abandoned the tank on its own will. Alternatively, $C$ may have cheated $P$ by having made no deal with $T$. Thus, it may have abandoned the tank instead of delivering it to $T$. Finally, $T$ may have correctly received the tank. However, to cut costs, it may have abandoned it instead of neutralizing its contents.

Starting from this case study, we interviewed several organizations operating in the domain of logistics - with special emphasis on hazardous goods - and we identified the following requirements that a monitoring platform, in general, should meet:

- **R1**: Monitoring information should be collected limiting as much as possible the intervention of human operators as human operators are prone to make mistakes and misleading information could be introduced.
- **R2**: The monitoring platform should not expect the process to be executed as initially agreed. Otherwise, deviations in the process would not be captured.
- **R3**: The monitoring platform should not stop when a discrepancy between the expected behavior and the observed one is detected. Otherwise, subsequent deviations would not be captured.
- **R4**: Monitoring information should be made available to all the organizations participating in the process, either directly or indirectly (i.e, as an observer). If monitoring information is not shared, the process could be monitored only partially. Alternatively, a central entity – trusted by all the participants – should be responsible for monitoring the process. All the participants should be aware on how the whole process, and not only their portions, will be carried out.
- **R5**: Monitoring information should be consistent and sent timely to all the participants. If different organizations have different monitoring information, then it would be difficult to know who has the right information.
- **R6**: If needed, organizations that did not take part in the process (e.g., a public prosecutor) should be allowed to access monitoring information even after the process completed.
- **R7**: Nobody should be able to alter monitoring information. Otherwise, if an organization incorrectly performed part of the process, it may alter monitoring information either to blame somebody else or to prove that they were compliant.
- **R8**: Nobody should be able to send monitoring information on behalf of somebody else. Otherwise, an organization may impersonate another one and send fraudulent information in order to blame somebody else.

The artifact-driven monitoring approach [9] can address some of these requirements. In fact, artifact-driven monitoring detects when activities are executed based on the conditions of the physical objects (i.e., artifacts) in the process. For example, the conclusion of activity Attach tank to truck can be detected when the artifact Tank is full and attached to the truck. Therefore, if these objects are made smart with the IoT paradigm, they can autonomously provide this information, thus addressing *R1*. Also, by relying on the artifact-centric language Extended-GSM (E-GSM) to represent the monitored process, artifact-driven monitoring can transparently deal with variations in the control flow, thus addressing *R2* and *R3*. For example, by modeling the process presented in Section 2 in E-GSM, control flow dependencies are considered descriptive rather than prescriptive. This way, artifact driven monitoring can detect – even though such an occurrence violates

the control flow dependencies – if the tank was leaking and $P$ did not emptied it. For further details on E-GSM and the advantages it provides for monitoring, we invite the reader to consult [2]. Finally, an artifact-driven monitoring platform can run entirely on top of physical objects exchanging monitoring information with each other, thus addressing *R4*.

However, as artifact-driven monitoring was not designed with trust in mind, it presents some limitations in this regard. Firstly, monitoring information is stored in the memory of each smart object. Consequently, anyone who has physical access to a smart object can potentially alter monitoring information and make it inconsistent. Secondly, no mechanism to verify either the origin or the correctness of monitoring information exchanged by smart objects was put in place. Therefore, any smart object that participates in the process can send monitoring information on behalf of any other smart object. Also, a compromised smart object may send monitoring information that does not reflect the actual state of the smart object. Finally, an artifact-driven monitoring platform relies on a centralized message bus to distribute monitoring information to the smart objects. Thus, a failure on that component would prevent the monitoring platform to correctly work.

## 3 Adopting blockchain to improve the trust in monitoring

To implement a fully trusted monitoring platform, thus addressing *R5*, *R6*, *R7*, and *R8*, we investigated the possible adoption of a blockchain. A blockchain [13] is a distributed ledger in which information is stored in a safe, verifiable and permanent way. Every time a new piece of information has to be made available to other participants, a new transaction is created. Transactions are then grouped into a block that references the previously stored block, hence the name "blockchain". Once a new block is created, it is validated and then made available to all the participants in the blockchain. This mechanism allows a blockchain to provide the following features:

- **Distributed consensus**: Multiple participants are responsible for validating information written on a blockchain. Therefore, as long as the validation mechanism is correctly implemented, it is impossible for a single participant to introduce incorrect information. Also, when a new participant joins the blockchain, it can obtain its own copy of the blockchain and participate in the validation process. This way, *R5* and *R6* can be addressed.
- **Persistence**: Each block in the blockchain directly references the previous one via a hashing mechanism. Therefore, it is impossible for a single participant to alter or delete a transaction once it has been written in a block. In addition, as the validation mechanism requires information on the previous blocks, every participant that performs this task holds a complete copy of all the data stored in the blockchain. Thus, even if a participant loses its copy of the blockchain, multiple copies of the data are still available, and *R7* can be addressed.
- **Auditability**: When a new transaction is created, it contains a timestamp and a signature identifying the participant who created it. Therefore, it is

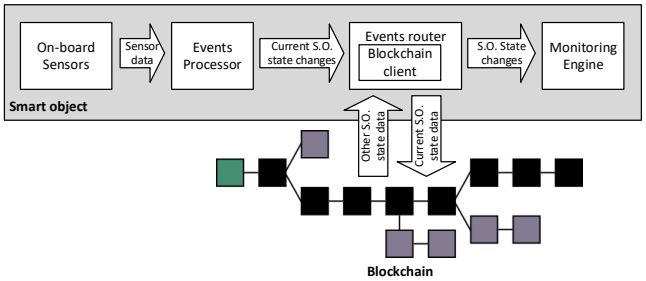


Fig. 2: Architecture of the fully blockchain-based platform.

impossible for a participant to create a transaction on behalf of another one. This way, *R8* can be addressed.

To provide a fully trusted monitoring solution, in sections 3.1 and 3.2 we present the architecture of two alternative platforms that, respectively, treat the enforcement of persistent monitoring information and the reduction of the information written on-chain as first-class citizens. In particular, the first platform fully guarantees that monitoring information will never disappear in exchange for a variable computational effort to be performed on-chain. Instead, the second platform makes the amount of information written on the blockchain independent on the amount of monitoring information produced. However, it does not enforce the persistence of monitoring information.

### 3.1 Fully blockchain-based platform

The first platform relies entirely on a blockchain to store and forward monitoring information. As shown in Figure 2, we take the architecture of an artifact-driven monitoring platform as a starting point. In particular, the physical characteristics of a smart object are collected by *On-board Sensors* and, thank to the *Events Processor* module, they are discretized into a finite set of states representing its conditions. Then, changes in the state of the smart objects participating in the process are notified to the *Monitoring Engine* module, which detects when activities are executed and identifies possible violations.

However, to let smart objects exchange information on their current state, we re-engineered the *Events Router* module, which implements information exchange mechanisms and policies, by integrating it with a *Blockchain Client*. The *Blockchain Client* is responsible for initiating a new transaction whenever the current smart object changes state, and for sending a notification to the *Monitoring Engine* whenever a new block containing a transaction from the other smart objects is added to the blockchain. Each smart object acts as a specific participant, and it has a unique address.

Since a blockchain does not guarantee that monitoring data are recorded in the same chronological order as when they are produced, a reorder buffer

```
1  contract Blockclient {
2    string processModel; //process model
3    struct State { //event
4      uint id;
5      address sender;
6      string artifact;
7      string status;
8      string timestamp;
9      string data; }
10   mapping(uint => State) public states; //list of events
11   uint stateCounter;
12   struct participant {
13       bytes32 encodedArtifact; }
14   mapping(address => participant) public participants; //participants
15
16   function Blockclient(string _processModel, address[] _addrs, bytes32[]
         _encodedArtifacts) payable public {
17     for (uint p = 0; p < _addrs.length; p++) { //add participants
18       participants[_addrs[p]].encodedArtifact = _encodedArtifacts[p]; }
19     processModel = _processModel;} //store process model
20
21   function writeState(string _artifact, string _status, string _timestamp,
         string _data) payable public {
22     if (participants[msg.sender].encodedArtifact == stringToBytes32(_artifact
           ) { //check identity of sender and ownership of artifact
23       stateCounter++; //increment state counter
24       states[stateCounter] = State(stateCounter, msg.sender, _artifact,
             _status, _timestamp, _data); //store state data
25       LogWriteState(stateCounter, msg.sender, _artifact, _status, _timestamp,
               _data); }}} //emit a new event
26
27   function getProcessModel() public view returns(string) {
28     return processModel;} //retrieve process model
29 }
```

Fig. 3: Excerpt of smart contract supporting the fully-blockchain based platform.

has been introduced in the *Events Router*. In fact, transactions can be stored in reverse chronological order if the later ones are included into a block before the earlier ones are validated. Consequently, if the chronological order is not respected, the *Monitoring Engine* may incorrectly monitor the process. By buffering transactions until five subsequent blocks have been written on the blockchain, and then reordering them based on the timestamp when data were collected by sensors, it is possible to minimize the occurrence of monitoring errors caused by transactions violating the chronological order of monitoring data.

Before the process starts, the smart contract shown in Figure 3 is deployed on the blockchain[1]. This smart contract contains the serialized model of the process to monitor (line 2) and a list of the smart objects participating in it (lines 12-14), which are instantiated once the contract is deployed (lines 16-19). It also defines a data structure to store monitoring information (lines 3-11), as well methods to append new information (writeState, lines 21-25) and to retrieve the

[1] Smart contracts are implemented in Solidity. However, they can be easily ported to other languages.

serialized process model (getProcessModel, lines 27-28). In particular, monitoring information is represented as a mapping (line 10), whose items (lines 4-9) contain an unique identifier, the identity of the smart object communicating the change, the type of smart object, the state currently assumed by the smart object, a timestamp indicating when it changed state, and the sensor data used to infer the state.

Once the smart contract is deployed, the *Blockchain Client* of all the smart objects referenced in that contract configures the *Monitoring Engine* with the provided process model. In addition, the *Blockchain Client* subscribes to the LogWriteState, which is emitted whenever new monitoring information is added to the blockchain (line 25). When the *Events Processor* of one of these smart objects detects a change in its state, the *Blockchain Client* invokes the writeState method by passing the monitoring information, thus initiating a new transaction. The other participants in the blockchain validate the transaction, which is then written in a new block. Meanwhile, if the *Blockchain Client* is notified about a new occurrence of LogWriteState, it extracts the smart object and the new state assumed by it and forwards this information to the *Monitoring Engine*.

It is worth noting that every participant that joins the blockchain can validate transactions from the smart objects and collect monitoring information. This allows third parties, such as external auditors, to monitor the process. When some information, especially on the structure of the process and on the state assumed by the smart objects, should not be publicly disclosed, it is still possible to encrypt it by putting in place a Public Key Infrastructure (PKI) and traditional key distribution mechanisms. This way, only entitled participants can read this information, and confidentiality can be guaranteed.

In addition, monitoring information can be easily accessed after the process finished, even if the smart objects are no longer present. This makes possible for entitled third parties to determine how the process was performed simply by replaying monitoring information. For example, still referring to the case study, the authorities can easily identify the organization responsible for having improperly disposed the tank, even if its memory was damaged. In fact, authorities can simply query the blockchain to obtain the process and all the changes in the state of the smart objects, being sure that this information was not altered once it was written on the blockchain. Then, they can instruct a *Monitoring Engine* with the E-GSM model, and replay the state changes to detect which portion of the process was incorrectly executed.

The main disadvantage of this platform is the potentially high amount of information that is written on the blockchain, which limits its applicability in conjunction with a public blockchain. Since a public blockchain requires a fee to be paid for each invocation of the smart contract, and the fee is dependent on the amount of information that is passed, the more information has to be written, the more expensive the monitoring platform will be. For example, as long as the state is simply represented by a label, the cost will be quite low. However, if the participants would like to know how the state was determined, then also the historical values collected by sensors and the rules adopted to derive the state

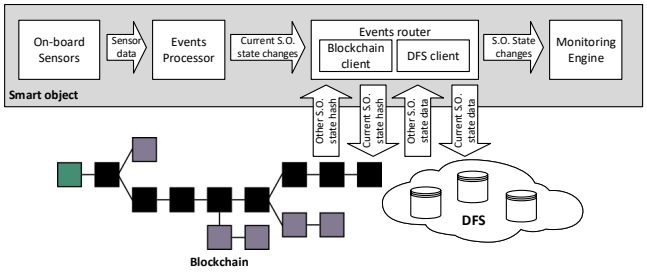


Fig. 4: Architecture of the DFS-blockchain hybrid platform.

should be written on the blockchain, thus significantly increasing the cost of the platform.

### 3.2 DFS-blockchain hybrid platform

To address the limitations of the fully blockchain-based platform, the cost associated to each transaction must be reduced and, possibly, made independent of the amount of information that is written. To this aim, we propose a second platform that relies both on a blockchain and on a publicly accessible Distributed File System (DFS). Like in the fully blockchain-based platform, this one relies on a blockchain to notify the smart objects on the process to monitor and on changes in their state. However, monitoring information is not stored inside the blockchain. Instead, information is stored as a file in the DFS, and only a reference to the file is stored in the blockchain.

To support this platform, the reference architecture shown in Figure 4 is proposed. In this case, the *Events Router* integrates both a *Blockchain Client* and a *DFS Client*. Like in the previous architecture, the smart contract shown in Figure 5 is deployed on the blockchain before the process starts (portions identical to the one shown in Figure 5 are omitted). However, this smart contract does not directly contain the serialized process model (line 2). Instead, the process model is stored in the DFS as a file, and a hash of that file is computed and stored in the smart contract. This way, once the smart contract is deployed, the *Blockchain Client* retrieves the hash of the process model by invoking the getProcessModelHash method. Then, it obtains the process model by asking the *DFS Client* to retrieve the file whose hash matches the one specified in the smart contract and, once received, it configures the *Monitoring Engine*.

Similarly, the data structure in this smart contract (lines 3-8) does not store the states assumed by the smart objects. Instead, it stores the hash computed from this information. This way, when the *Events Processor* detects a change in its state, the *Blockchain Client* asks the *DFS Client* to write the new state in a file, then it invokes the writeHash method by passing the hash of the newly created file, thus initiating a new transaction. Then, once the *Blockchain Client* is notified on a change in the state of a smart object (event LogWriteHash), it

```
contract IPFSblockclient {
  string processModelHash;
  struct StateHash {
    uint id;
    string mHash;
  }
  mapping(uint => StateHash) public hashes;
  uint hashCounter;

  function IPFSblockclient(string _processModelHash, address[] _addrs,
      bytes32[] _encodedArtifacts) payable public {
    for (uint p = 0; p < _addrs.length; p++) { ... } //add participants
    processModelHash = _processModelHash; } // store process model hash

  function writeHash(string _artifact, string _mHash) payable public {
    if (participants[msg.sender].encodedArtifact == stringToBytes32(_artifact
        )) { //check identity of sender and ownership of artifact
      hashCounter++; //increment state counter
      hashes[hashCounter] = StateHash(hashCounter, _mHash); //store state
          hash
      LogWriteHash(hashCounter, _mHash); }}} //emit a new event

  function getProcessModelHash() public view returns(string) {
    return processModelHash;} //retrieve process model hash
}
```

Fig. 5: Excerpt of smart contract supporting the DFS-blockchain hybrid platform.

receives the hash of the new state and then asks the *DFS Client* to retrieve the file matching that hash. Since the hash computed for each invocation of the smart object has a fixed length, the cost of each transaction is independent from the amount of data that is produced. Therefore, information on the state of a smart object can be enriched with historical sensor data and discretization rules without increasing the cost of the transaction. This allows to implement more sophisticated (off-chain) validation mechanisms, to ensure that monitoring information was not originated by a faulty or compromised smart object.

As in the case of the fully blockchain-based one, this platform guarantees the immutability of monitoring information. In fact, altering monitoring information would require one or more files stored in the DFS to be changed. However, a minimal modification in a file would completely change its hash, preventing it to be retrieved by the other participants. As files are not stored in a central location, and each file can be replicated an arbitrary number of times, decentralization is also guaranteed. However, this solution does not enforce persistence of monitoring information by design. In fact, unlike a blockchain, a DFS does not force participants to have a copy of all the stored information. Therefore, unless data retention policies are enforced by the organizations, nobody prevents the participants from deleting information stored inside the DFS once the process ends. Thus, if nobody keeps a copy of this information, it is lost.

| *Platform* | *Enforces distributed consensus* | *Enforces persistence* | *Enforces auditability* | *Permissioned blockchain* | *Public blockchain* |
|---|---|---|---|---|---|
| **Fully BC-based** | Yes | Yes | Yes | Yes | Not recommended |
| **DFS-BC hybrid** | Yes | Not automatically | Yes | Yes | Yes |

Table 1: Comparison of the two proposed platforms.

### 3.3 Comparision

Table 1 summarizes the most significant aspects of the two proposed platforms. In particular, both platforms enforce distributed consensus and auditability, and they both can be easily implemented using a permissioned (i.e., private) blockchain, which does not require a fee to be paid for each transaction. Only the fully blockchain-based platform automatically enforces persistence of monitoring information, as it stores this information on the blockchain. However, this increases the operational cost of the solution when it operates on a public blockchain, making such a combination not recommended.

## 4 Validation

To demonstrate the applicability of our solution on a real-world use case, we built a prototype of the two platforms described in Section 3[2] starting from the source code of the SMARTifact platform [1]. For the implementation of the Events Router module, we chose the Ethereum [17] blockchain and the InterPlanetary File System (IPFS)[3] DFS due to both the availability of several tools, libraries, and testing infrastructures (e.g., testnets). In addition, a node implementing an Ethereum client can participate both in a private instance of the Ethereum blockchain and in the public one (i.e., the mainnet). As in the case of the original SMARTifact platform, also this one was entirely run on a Single Board Computer (SBC), in this case a Raspberry Pi[3]. To reduce the workload on the SBC, we configured the *Blockchain Client* as an Ethereum lightweight node. Lightweight nodes do not execute smart contracts, validate transactions, or require a complete copy of the whole blockchain to be downloaded. Therefore, their computational and storage requirements are low enough to be fulfilled by an SBC.

We then tested the two prototypes against a dataset provided by an European logistics company[4], which was also used in [10] to validate artifact-driven monitoring. This dataset contains: *(i)* a model of 6 delivery processes with a total of 53 execution traces; for each execution, *(ii)* logs containing the position and the speed of the involved trucks. Following the artifact-driven approach, we enriched the process model by associating to each activity a finite set of states representing the conditions on the truck required for the activity to start or finish.

[2] Source code at https://bitbucket.org/polimiisgroup/ethereumclient.
[3] http://www.raspberrypi.org
[4] The (anonymized) dataset is available at http://purl.org/polimi/martifact/logisticsds-anon (password: GM-CDC-JM-dataset).

| *Process name* | *Executions per process* | *Correctness (%)* | *Completeness (%)* | *Median transactions per execution* | *Contract deployment cost (gas)* | *Median cost per transaction (gas)* | *Median cost per execution (gas)* |
|---|---|---|---|---|---|---|---|
| **AMS-BRU** | 9 | 100.00 | 77.78 | 5.67 | 3276717 | 724547 | 4472261 |
| **AMS-CDG** | 8 | 100.00 | 87.50 | 8.88 | 3298198 | 724611 | 6846820 |
| **AMS-FRA** | 4 | 75.00 | 100.00 | 10.75 | 3277485 | 724529 | 8608058 |
| **AMS-LHR** | 12 | 91.67 | 58.33 | 10.58 | 3766963 | 724564 | 7979801 |
| **BRU-AMS** | 10 | 90.91 | 90.91 | 5.80 | 3298710 | 724609 | 4532603 |
| **CDG-AMS** | 10 | 100.00 | 60.00 | 11.00 | 3298710 | 724486 | 8299217 |

Table 2: Results of the validation for the fully blockchain-based platform.

| *Process name* | *Executions per process* | *Correctness (%)* | *Completeness (%)* | *Median transactions per execution* | *Contract deployment cost (gas)* | *Median cost per transaction (gas)* | *Median cost per execution (gas)* |
|---|---|---|---|---|---|---|---|
| **AMS-BRU** | 9 | 100.00 | 77.78 | 5.67 | 1155343 | 116235 | 787424 |
| **AMS-CDG** | 8 | 100.00 | 87.50 | 8.88 | 1155343 | 116235 | 1176585 |
| **AMS-FRA** | 4 | 75.00 | 100.00 | 10.75 | 1155343 | 116235 | 1538362 |
| **AMS-LHR** | 12 | 91.67 | 58.33 | 10.58 | 1155343 | 116235 | 1326045 |
| **BRU-AMS** | 10 | 90.91 | 90.91 | 5.80 | 1155343 | 116235 | 789697 |
| **CDG-AMS** | 10 | 100.00 | 60.00 | 11.00 | 1155343 | 116235 | 1394119 |

Table 3: Results of the validation for the DFS-blockchain hybrid platform.

Then, we defined rules to derive the state of the truck from logs, in order to autonomously monitor the process. Finally, we configured both platforms with the model and rules, and we replayed logs simulating the actual execution of the process. A new transaction was initiated every time a rule detected a change in the state of the truck. The transaction contained the new state, together with the most recent changes in the position and speed of the truck, amounting of 800 Byte of data. The prototype was tested with both a private instance of the Ethereum blockchain, and the Rinkeby public testnet[5]

Tables 2 and 3 summarize the results of the test using the fully blockchain-based and the hybrid platform, respectively. With respect to the artifact-driven monitoring platform described in [10], both platforms were capable of correctly monitoring the same process instances, as shown in columns *correctness* and *completeness*. Therefore, requirements *R1*, *R2*, *R3*, and *R4* were satisfied by both platforms. Also, thank to smart contracts, both platforms were able to satisfy requirements *R5*, *R6*, *R7*, and *R8* as well. Based on the complexity of the smart contract, i.e., the amount of data that has to be written and the operations performed on the data, a value, named *gas*, is determined for each transaction. It is worth noting that the DFS-blockchain hybrid platform requires less than a third of the gas required by the fully blockchain-based platform. In fact, monitoring a process execution with the former requires between 787424 and 1538362 gas units, while with the latter it ranges between 4472261 and 8608058 gas units. In addition, every transaction initiated by the hybrid platform will store on the blockchain always the same amount of data, corresponding to the hash of the monitoring information. Therefore, the gas units required per transaction will

[5] http://www.rinkeby.io

always be the same, regardless of the amount of monitoring information that has to be stored. On the other hand, the amount of information written on the fully blockchain-based platform equals to the monitoring information collected by the platform. Therefore, if the amount of information generated per transaction will increase, the gas units per transaction will increase as well. It is worth noting, as already discussed in the previous section, that this higher gas requirement for the fully blockchain-based platform is compensated by the guarantee of persistence.

To use our prototype in conjunction with the public Ethereum blockchain, a fee directly proportional to the amount of gas consumed by the transaction has to be paid. However, the *gas price*, that is, the fee per gas unit, is not fixed and it can be defined when the transaction is initiated. In general, the higher the gas price is set, the faster the transaction will be processed. When carrying out the experiment, we had to set the gas price to 5 GWei ( $5 \times 10^{-9}$ Ether), that is, circa $5.9 \times 10^{-7}$ €[6]. As a consequence, the operational cost of the fully blockchain-based platform would range between 2.64 € and 5.08 € per process execution, and the one of the hybrid platform would range between 0.46 € and 0.91 €. Such a difference in terms of cost is even more pronounced if we consider larger and more complex processes, such as the ones included in the datasets of the 2014-2015-2017-2018 BPI Challenges[7]. For each process execution in these datasets, the number of transactions is on average up to six times the one considered in our dataset[8]. Thus, for the fully blockchain-based platform, the maximum cost would be at around 30 € per process execution, whereas for the DFS-blockchain hybrid platform it would stay under 5 € per process execution.

Such an high price makes reasonable to adopt the public Ethereum blockchain only in conjunction with the DFS-blockchain hybrid platform, and only when processes manipulating very dangerous (e.g., nuclear waste) or highly valuable goods have to be monitored. In the other cases, a private instance of the Ethereum blockchain, internally used by the participants and which does not require any fee to be paid, is probably more advisable. However, we expect the upcoming introduction of the proof-of-authority consensus algorithm in the public Ethereum blockchain to sensibly decrease the operational cost. In fact, proof-of-authority will significantly decrease the computational effort required to generate a new block, causing the high value of gas price to no longer be justified. In such a scenario, gas price would drop and, consequently, a monitoring platform relying on the public Ethereum blockchain would become affordable also for general purpose business processes.

## 5 Related Work

Traditionally, business-to-business communications have been performed with the Electronic Data Interchange (EDI) standard [6]. However, EDI has been conceived with commercial transactions in mind. In addition, it requires participants to join

[6] The conversion rate was checked on March 15, 2019.

[7] https://www.win.tue.nl/bpi/doku.php?id=2018:challenge

[8] We assumed a transaction to be initiated every time an event is produced.

a dedicated commercial network, which requires participants to pay a subscription in order to be admitted. Finally, EDI does not implement any mechanism to archive transactions, nor to certify the identity of the sender. To improve trust in information systems, the adoption of blockchain has been investigated as a valuable solution [14,16]. More specifically, [8] presents an exhaustive analysis of the implications of introducing a blockchain in inter-organizational processes and, among the others, the need for developing a diverse set of process monitoring frameworks on a blockchain. Also, [7] outlines the potential advantages of the synergy between blockchain and business artifacts.

To this aim, [12] exploits the Bitcoin blockchain to monitor and verify process choreographies. Starting from a BPMN collaboration diagram, a set of smart contracts is derived. Similarly, [5] proposes an approach to derive smart contracts from multi-party processes modeled as Petri nets, which is validated with respect to cost. [15] proposes supply chain traceability system relying on blockchain and on the IoT. Finally, [18] proposes a framework for coordinating and monitoring transportation processes based on several private blockchain installations, globally managed by a public blockchain. However, none of these works allow deviations in the execution order of activities. Consequently, the execution of activities that do not follow control flow dependencies is not detected. In addition, neither [12] nor [5] take into consideration the conditions of the physical objects (i.e., the artifacts) participating in the process. Such conditions are useful to determine if an event has occurred for real or it has been incorrectly reported.

## 6 Conclusions and Future Work

This paper presented how to provide a trusted monitoring platform for multi-party business processes. Starting from the benefits of the artifact-driven monitoring approach, the impact of blockchain adoption to provide a trusted environment has been analyzed. In particular two configurations of the proposed platform have been implemented and their pros and cons have been evaluated with a set of experiments. The results show that a DFS-blockchain hybrid platform is significantly less expensive than a fully blockchain-based one. Nevertheless, the second is preferable when the persistence of monitoring information has to be enforced.

A limitation of this approach consists in relying on off-chain software modules, like the *Events Processor* and the *Monitoring Engine*, to monitor the process. Consequently, the E-GSM model and the rules to determine the state of the smart objects cannot be formally validated by the blockchain. To improve this situation, future work will investigate the adoption of oracles to ensure the correct execution of these modules. Also, although the adoption of blockchain has the merit of increasing the trust in monitoring, the proposed solutions do not provide any type of control for possible malicious modification of the data before they are sent to the blockchain. For this reason, tamper-proof systems must be considered in the up-link, i.e., between the sensors and the chain. At the same time, the adoption of a blockchain brings the current disadvantages of this technology in

terms of performances. In fact, writing, approving, and distributing a new block to all the participants takes seconds for a permissioned blockchain, or even several minutes for a public one. Nevertheless, research efforts to speed up operations on a blockchain are currently being taken by both academics and the industry, so we expect this issue to be eventually solved or scaled back.

## Acknowledgments

This work has been funded by the Italian Project ITS Italy 2020 under the Technological National Clusters program, and by the DITAS project funded by the European Union's Horizon 2020 research and innovation programme under grant agreement RIA 731945.

## References

1. Baresi, L., Di Ciccio, C., Mendling, J., Meroni, G., Plebani, P.: martifact: an artifact-driven process monitoring platform. In: BPM Demo 2017. Springer (2017)
2. Baresi, L., Meroni, G., Plebani, P.: Using the guard-stage-milestone notation for monitoring bpmn-based processes. In: BPMDS EMMSAD 2016. pp. 18–33. Springer
3. Benet, J.: Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561 (2014)
4. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer (2018)
5. García-Bañuelos, L., Ponomarev, A., Dumas, M., Weber, I.: Optimized execution of business processes on blockchain. In: BPM 2018. pp. 130–146. Springer (2017)
6. Hsieh, C., Lin, B.: Impact of standardization on EDI in B2B development. Industrial Management and Data Systems 104(1), 68–77 (2004)
7. Hull, R., Batra, V.S., Chen, Y., Deutsch, A., III, F.F.T.H., Vianu, V.: Towards a shared ledger business collaboration language based on data-aware processes. In: ICSOC 2016. pp. 18–36. Springer (2016)
8. Mendling, J., et al.: Blockchains for business process management - challenges and opportunities. ACM Trans. Manage. Inf. Syst. 9(1), 4:1–4:16 (Feb 2018)
9. Meroni, G., Baresi, L., Montali, M., Plebani, P.: Multi-party business process compliance monitoring through iot-enabled artifacts. Inf. Syst. 73, 61–78 (2018)
10. Meroni, G., Ciccio, C.D., Mendling, J.: An artifact-driven approach to monitor business processes through real-world objects. In: ICSOC 2017. pp. 297–313. Springer (2017)
11. Meroni, G., Plebani, P.: Combining artifact-driven monitoring with blockchain: Analysis and solutions. In: CAiSE 2018 Workshops. pp. 103–114 (2018)
12. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime verification for business processes utilizing the bitcoin blockchain. FGCS (2017)
13. Swan, M.: Blockchain: Blueprint for a new economy. " O'Reilly Media, Inc." (2015)
14. Tai, S.: Continuous, trustless, and fair: Changing priorities in services computing. In: Lazovik, A., Schulte, S. (eds.) ESOCC 2018. pp. 205–210. Springer (2018)
15. Tian, F., Taudes, A., Mendling, J.: A supply chain traceability system for food safety based on haccp, blockchain amp;amp; internet of things. In: ICSSSM 2017. pp. 1–6. IEEE (2017)

16. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: BPM 2016. pp. 329–347. Springer (2016)
17. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151, 1–32 (2014)
18. Wu, H., Li, Z., King, B., Ben Miled, Z., Wassick, J., Tazelaar, J.: A distributed ledger for supply chain physical distribution visibility. Information 8(4), 137 (2017)