A Lean Architecture for Blockchain Based Decentralized Process Execution

Christian Sturm[1], Jonas Szalanczi[1], Stefan Schönig[1], and Stefan Jablonski[1]

Universität Bayreuth
{firstname.lastname}@uni-bayreuth.de

Abstract. Interorganizational process management bears an enormous potential for improving the collaboration among associated business partners. A major restriction is the need for a trusted third party implementing the process across the participating actors. Blockchain technology can dissolve this lack of trust due to consensus mechanisms. After the rise of cryptocurrencies, the launch of *Smart Contracts* enables the *Ethereum Blockchain* to act beyond monetary transactions due to the execution of these small programs. We propose a novel lean architecture of a Blockchain based process execution system with Smart Contracts to dispense with a trusted third party in the context of interorganizational collaborations.

Keywords: Business Process Management · Blockchain · Collaborative Process Management · Choreography Processes · Process Execution

1 Introduction

Blockchain technology currently triggers a revolution in the way we store our data: data move from centralized (cloud) storage towards decentralized data management and information systems. The technology has proven to bear a great potential for disruptive change in many domains. Our aim is to exploit and leverage on this technology when organizational processes in context of Business Process Management has to be enacted.

Blockchain technology became famous as backbone behind the cryptocurrency Bitcoin [12], followed by a vast number of alternative cryptocurrencies. While these 1st-generation Blockchains were originally focused on monetary transactions, next-generation Blockchains like *Ethereum* were established further on [3]. The latter provides the turing-complete programming language *Solidity* on top for executing small programs, called *Smart Contracts* directly on the Blockchain. The idea of Smart Contracts was first described 1997 in [17] and was resurged with the Ethereum Blockchain where Smart Contracts are account holding objects following several principals: They provide code functions, they can interact with other contracts or users and they are able to make decisions based on stored data.

The key fundamental concept of using a Blockchain instead of legacy systems is the tamper-proof character of the underlying database (distributed ledger)

without the need of having a trusted third party included. Different applications had proven the feasibility of Smart Contracts ranging from government regulations regarding manufacturing of pharmaceuticals [13] to organizing rescue packages in crisis areas [16].

With respect to Business Process Management, especially the collaboration between companies through choreography processes seems a suitable and profitable application domain. Unlike orchestration processes in inter-organizational process management, where the participants are also owner of the processes, we assume an adversarial setting where participants may blame each other as pointed out in [10]. The consensus mechanism in Blockchain technology is said to annul the need for a trusted third party as an arbitrator, because the nodes in the network will reach accordance by their selves and thus are impervious to potential attackers.

Our contribution composes of a novel architecture using Blockchain technology as a backbone for inter-organizational process execution. The lean architecture enables a lightweight full-featured on-chain implementation of a decentralized process execution system. We exploited the latest advanced concepts of the Solidity programming language for our Smart Contract. Compared to existing approaches, the omission of additional software artefacts in our architecture leads to an entire on-chain execution. For large process models comprising a big number of tasks, we believe that our approach scales best. With our solution, we state that most of the execution steps should be secured on the Blockchain. We further critically analyse our approach, compare it with existing alternatives, and discuss advantages and disadvantages.

The remainder of the paper is structured as follows. In Section 2 we provide a comprehensive overview of the principles, how Blockchain technology works internally. Then we summarize Related Work of using Blockchain technology in Business Process Management in Section 3. Our approach is the main constituent of Section 4 and after a qualitative Evaluation in Section 6, we conclude the paper in Section 7.

2 Background

This section provides an overview on the most important concepts of Blockchain technology. We describe, how transactions are validated and how consensus can be reached and point out differences between a public Blockchain and a Consortium Blockchain. The main reason that different branches in industry investigate Blockchain technology is the non-necessity of a trusted third party. In contrast, traditional execution of financial transactions between two parties always requires the bank of the sender as well as the bank of the receiver as intermediary players. We refer to [1] for more detailed information.

The backbone of the Blockchain, a linked list of blocks comprising transactions, is a Peer-to-Peer network of participants, so-called *(full) nodes*. Each of them holds the entire Blockchain locally including all transactions ever processed. A new transaction is propagated within the whole network and each node

includes it into its pool of unconfirmed transactions. Always, nodes try to *mine* a new block by solving a cryptographic puzzle. All unconfirmed transactions in the pool as well as additional information (ID of the latest block, the timestamp, an adjustable *nonce*) serves as input for a hashing algorithm. The nonce is generated at random, until the output of the hashing algorithm falls below a certain limit. If such a nonce is found, the node wins this laborious mining game and informs the network. The unconfirmed transactions that were affecting the hash output are confirmed, and all nodes remove them from their pools. As a reward, the miner receives the fees associated with the transactions. The confirmation of transactions relies on the principle, that every node can double-check a new propagated block against a set of specific rules. Hence, if invalid transactions, e.g. fraudulent double spends of monetary units, are included in the new block, the network rejects it and miners would waste a vast amount of computations for finding an expedient nonce and thus waste electricity and thus money. Additionally, also the mining fees are lost.

The term *Blockchain* is often referred to the public Blockchain, which is open and free to access to anyone willing to participate in the P2P network. This may cause trouble regarding privacy especially when storing data payload on-chain. As an alternative, private or Consortium Blockchains are isolated from the global network and thus read/write accesses can be permissioned and adjusted. Consortium Blockchains are often confused with private Blockchains, but the latter denies the participation of external actors whereas the former just restricts the permission to mine and verify. In private Blockchains, mining and so consensus power is much more concentrated than in public Blockchains. Therefore, one has to pay attention that a participant of the consortium does not gain too much power. Janne Hansen, IT-architect and developer from Microsoft, states that at least four parties should form a Consortium Blockchain for reasons of trust[1]. Some more differences to public Blockchains and the reasons why we rely rather on private or Consortium Blockchains in our context are discussed further in Section 6. However, the proposed architecture is not limited in the type of the underlying Blockchain.

3 Related Work

In [11], the authors emphasize the chances of using Blockchain Technology in the context of Business Process Management. They provide a broad overview of open research challenges w.r.t. the process life cycle. They also discuss critical issues of applying this technology in the area of Business Process Management, for instance network performance, security and usability. They further stated that inter-organizational processes are affected by a lack of mutual trust, which can be solved by providing a trustworthy environment through Blockchain technology. They believe that a system can support global process monitoring where the encryption mechanism is used to handle privacy. However, their work is rather conceptually and they do not focus on or provide any implementation aspects.

[1] https://jannehansen.com/where-consortium-blockchains-fit/

In [19] the feasibility of Blockchain technology for process management was proven for the first time. This work includes a **Translator** for an automated translation of BPMN process models to Smart Contracts as well as two different artefacts for the execution: a choreography monitor (**C-Monitor**) for passive monitoring and an active **mediator**. A **trigger** establishes a connection between the artefacts on the Blockchain and the external workflow management systems of the involved participants. The Smart Contract stores two lists, which are used for encoding the process status. Their system enables collaborative process execution over untrusted nodes where all transactions are stored immutable and only conforming cases are executable. They completely rely on executions on a public Blockchain and thus their architecture is affected by some restrictions. For instance, they propose that not all aspects of collaborative process management should be transferred onto the Blockchain due to costs for data storage, transactions and computation. On the other hand, they also had to implement triggers to connect the Blockchain technology world with the enterprise internal process engine, because Smart Contracts could not call external APIs. However, Solidity was developed further and supports function callbacks to the user-interface and error-handling in the meantime. Further on, they have to create a specific Smart Contract for each process model, whereas our Smart Contract is more generic. The work of [19] was revived in [5] and again highlights the need for resource usage optimization driven by the execution on a public Blockchain. This is done by detouring the translation of BPMN models to Smart Contracts via Petri Nets which are minimized. They also took care of the number of deployed contracts as well as throughput rates by optimizing runtime components.

The authors of [8] adopted the approach of artefact-driven business modelling and make use of Blockchain technology for a shared ledger Business Collaboration Language. They discussed the advantages of domain specific languages on a rather conceptual level without any implementation aspects regarding Smart Contracts. [9] introduces Caterpillar, a Business Process Management System that runs on top of the Ethereum Blockchain and makes use of the work in [19], for instance the BPMN-to-Solidity translator. The work does not give enough insights how the internal structures are build and work together or how the system can be extended. Madsen et al. demonstrated in [10] a declarative workflow execution based on DCR-Graphs [6] in an adversarial setting in which Smart Contracts on Ethereum solves the lack of a trusted third party. They also had taken cost issues into concern due to the execution on the public Blockchain. In contrast to them, we see our approach not solely in adverse settings, but also where documentation and traceability plays a key role, for instance regarding quality management in the supply chain. The problem of the immaturity of the Ethereum Blockchain is discussed in [15] where the decision was made in favour of the Bitcoin Blockchain to address the shortcomings regarding major stability issues. However, this 1st-generation Blockchain restricts the opportunities by far. Hence, the authors focus rather on runtime verification of processes than on a sophisticated execution engine.

4 Lean Architecture with a Generic Smart Contract

We present a source-code optimised solution for executing business processes on the Blockchain. At the time of writing, our implementation comprises just around 100 lines of code (independent from the underlying process model) and yet, the deployed Smart Contract is able to run a fully - albeit rudimentary - system for collaborative process execution on the Ethereum Blockchain. Additionally, due to brevity, the contract is easy to maintain, to extend and to integrate via its *Application Binary Interface*. Contrary to the work of [19], we use a generic Smart Contract, i.e. our Smart Contract serves as scaffolding for each process instance and process model. Once deployed, the logic of the implemented process is poured into the Smart Contract by sending transactions (see below). Using this approach, also the process logic itself is on-chain and thus transparent and trustworthy, whereas in the architecture of [19] a superior authority creates and deploys a process-specific Smart Contract. We refer to this issue in Section 6.

Infrastructure of the Architecture. As a prerequisite, every participant who wants to join the collaboration, i.e. execute a task, has to establish a connection to the Ethereum Peer-to-Peer-network. This is done with the help of a so-called *wallet*, which assigns the user an unique identifier (160-bit hash value). The user can interact through this *account* with the Ethereum Blockchain, i.e. create transactions and interact with Smart Contracts. Consider Figure 1. A *Supervisor* S deploys as a first step the Smart Contract out of his wallet (1a) with a transaction (1b). After then, he can add users by registering the wallet addresses via addCollaborators (2) and create tasks in a similar way (3). Step (2) is optional but enables an advanced permission system (see below). The implementation of the core architecture is described in detail in Section 5. Our approach is easy to connect to organisation-internal structures. The Ethereum Virtual Machine allows callbacks and error-handling which can be used in a JavaScript interface on the client side. The role of the *Supervisor* can be assigned to any wallet and is responsible for the initial setup (e.g. addTasks) but the full transparency of the Blockchain does not limit the level of trust and tamper protection.

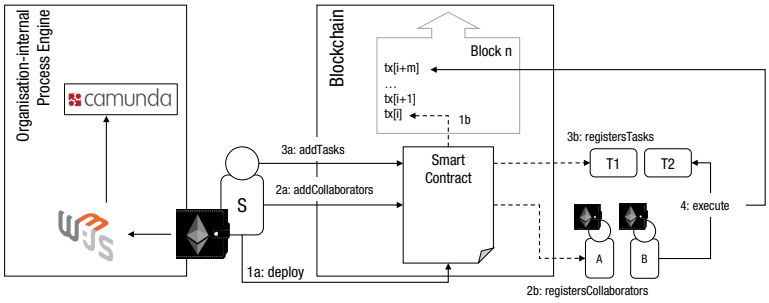

Fig. 1. Architecture of the Blockchain based process execution

Process Logic through Requirements. For the implementation of the process logic, the generic Smart Contract provides a struct Task. The struct contains a list requirements holding a reference to every task which must be set on completed before the next task can be completed.

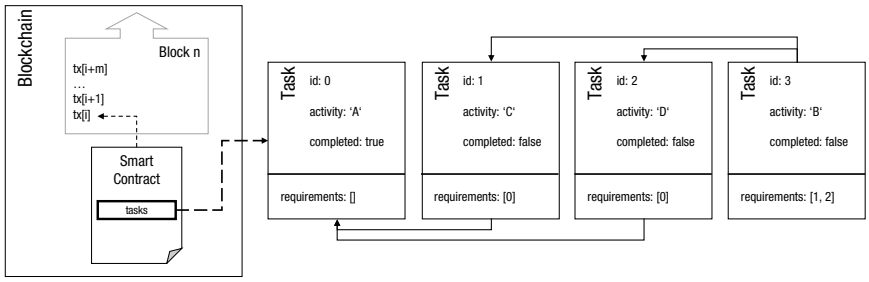

Fig. 2. Requirements

In the process model depicted in Figure 3 the task B have the requirements [A]. Referring to Figure 4, the Tasks C and D have [A] as a requirement and B has the requirements [C, D]. Figure 2 describes the situation after the execution of the activity A. C and D are now ready for execution. Based on the gateway, it is sufficient that at least one task is completed (*Exclusive Gateway (XOR)*) or both tasks in the requirements must be set on completed (*Parallel Gateway*). The Smart Contract implements the gateway logic as described in Section 5.

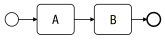

Fig. 3. Simple Sequence Flow

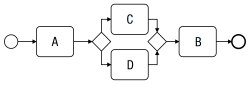

Fig. 4. Process Model with BPMN-Gates

Execution. For each process instance, one Smart Contract must be deployed on the Blockchain. After than, all collaborators (their wallet addresses) can be registered in the Smart Contract for access control purposes. To generalize the execution, the wallet addresses could be mapped organization-internal to a number of employees or IoT-devices etc. The next step is to build the process logic by registering the tasks, with the specific requirements. This is controlled by further transactions, i.e. function calls to the Smart Contract. This procedure can also be automated (cf. Section 5). To finish a task, another method call to the Smart Contract is required. To achieve conformance, this method checks if this task can be executed, i.e. if the requirements are fulfilled and if the user is allowed to complete the task.

5 Implementation

The architecture of the implementation is depicted in Figure 1. We only focus on the essential elements of BPMN to proof the feasibility of our approach. However, at spots where we omitted advanced concepts, we show up possible extensions we plan to implement in future.

From BPMN to the Smart Contract. We start from a BPMN process model which encodes the collaboration. The process model is parsed to find the requirements and generate the transactions. Contrary to [19], we do not generate a Smart Contract in this step. As stated, our generic Smart Contract is the same for every process (instance) at time of deployment, and is filled with logic by transactions. Thus, instead of creating Solidity programming code, the translator module outputs the transactions containing the requirements for instance.

The Smart Contract Scaffolding. One Smart Contract represents exactly one process model (or instance) and its structural elements are depicted in Listing 1 exemplary.

Listing 1: Structural concerns of the Collaboration Manager

```
1 contract ContractCollaborationManager {
2     address supervisor;
3     enum Tasktype {TASK, AND, OR};
4     struct Collaborator { address resource; string organisation; }
5     struct Task { string activity; address taskresource; bool completed;
        Tasktype tasktype; uint[] requirements; }
6     mapping(uint=>Task) tasks; uint[] public tasksArray;
7     mapping(uint=>Collaborator) collaborators; uint[] public collaboratorArray;
      }
```

The struct Collaborator defines the template for any organizational resource or actor, which is included in the process execution and potentially wants to execute a task. Currently, such a collaborator is described by address resource, the wallet address of the performer, and string organization that associates the resource with the employing organization. This can be used in future to restrict the execution not only to a specific person, but also to an organization in general instead. The execution permission mechanism (cf. Permission System) is described below.

The struct Task defines the template for any task in the process model. A task is identified by int id which is used in combination with the address taskresource field for checking the permission to execute the task or to store the actor which has actually executed the task. string activity gives a textual representation of the task and uint[] requirements empowers the conformance of the process execution (cf. Conformance System).

Note the behavioural concerns of the Smart Contract in Listing 2. Mainly two functions are responsible for the initialization or deployment of the Smart

Contract. Firstly, function addCollaborator to register every wallet address who wants to execute at least one task and secondly function addTask, to make the required tasks of the process model available. Note the third parameter of the function createTask. This enum can assume the values `TASK`, `AND` and `OR` and describes the represented BPMN element. This is an essential feature that is used in the Conformance System, described in Listing 3.

Listing 2: Behavioural concerns of the Collaboration Manager

```
1 contract ContractCollaborationManager {
2     function addCollaborator(address _collab, string _org) public { /*...*/ }
3     function createTask(string _activity, address _taskresource, Tasktype
       _tasktype, uint[] _requirements) public {
4         require(msg.sender == supervisor);
5         Task storage task = tasks[taskcount++];
6         task.taskresource = _taskresource;
7         task.requirements = _requirements;
8         /*...*/
```

Permission System. Our Permission System follows currently the guidelines described next, but is easy to adapt. The decision, who is permissioned to execute a task, is made at design time. When a new task is registered at the Smart Contract, the address _taskresource parameter (cf. function createTask in Listing 2, Line 6) specifies the sole wallet address which is permissioned to place the transaction of completing the task (with function setTaskOnCompleted, cf. Line 2 in Listing 3). This is essential when using the public Blockchain, because every wallet could interact with the Smart Contract. The relevant code can be refactored to address less restrictive policies for instance just a requirement that a registered Collaborator belongs to a specific organisation. Note that the organization is encoded in the struct Collaborator. A big advantage of using Blockchain technology is the inbuilt account system with wallet addresses to trace the transactions and interactions with their corresponding performer.

A different conceivable approach is to register a bunch of wallet addresses within the Smart Contract, so that the specific resource allocation can be done at runtime. The resource which wants to execute a task can log in with his credentials to a client-side user-interface which provides an assigned wallet address.

Conformance System. The Conformance System ensures, that the Smart Contract allows only valid process executions according to the initial BPMN model. Line 7 in Listing 2 assigns the ids of tasks which have to be completed, before a Collaborator wants to execute the next task with setTaskOnCompleted. The Conformance System is built upon the _requirements which are extracted from the BPMN model. For instance, in a trivial case the starting event with _id=0 has no requirement, i.e. it can be executed. The AND-gate is encoded in in Listing 3 from Line 8 on. The task is only set on completed (Line 12), when

Listing 3: Execution concerns of the Collaboration Manager

```
1  function setTaskOnCompleted(uint _id) public returns(bool success) {
2      require(tasks[_id].taskresource == msg.sender);
3      uint[] temprequire = tasks[_id].requirements;
4      /*Tasks */ if(tasks[_id].tasktype == Tasktype.TASK) {
5          if(isTaskCompletedById(temprequire[0]) == true) {
6              tasks[_id].completed = true; return true;
7          else { return false; }}
8      /*And-Gate */ if(tasks[_id].tasktype == Tasktype.AND) {
9          for(uint i = 0; i ¡ temprequire.length; i++) {
10             if(isTaskCompletedById(temprequire[i])==true) {tempcount++; } }
11         if(tempcount == temprequire.length) {
12             tasks[_id].completed = true; return true; }
13         else { return false; } /*...*/
```

all tasks in the requirements are completed. The contract uses a similar solution for the Or-gate, where only one of the requirements has to be fulfilled.

Advanced implementation concepts Contrary to Related Work that we evaluated, we were able to use language constructs which were not introduced back than to design a more sophisticated architecture.

One of the key concepts our proposed architecture is built on, is the require keyword which allows us to introduce a kind of *error-handling* and to have an interaction with the client side. Error-handling is one of the reasons, we could get rid of intermediary structures like triggers [19] to communicate with Smart Contract-external structures and therefore develop a much cleaner concept. We used require at several points, for instance to check if the current sender of a transaction is allowed to execute a task (cf. Listing 3, Line 2). When trying to complete an unassigned task, we receive an error message from the Ethereum Virtual Machine which we can handle in our external application. Also in this context, Solidity provides so-called events for enabling callbacks to a user interface. The concept of mapping helps us to set up a key-value data structure for all Tasks within the Smart Contract. In combination with this, the use of structs and enums gives us a customizable data structure and the final execution on the Smart Contract is according to that much more scalable (cf. Section 6). Due to lack of space, we provide the full Smart Contract at our GitHub-Repository[2].

6 Evaluation

In this section, we evaluate how our approach can solve the problem of the need for a trusted third party within collaborative process execution. We also enlarge on a performance and cost evaluation as well as on a comparison among related work.

[2] https://github.com/Jonasmpi/PExSCo

Solving the lack of trust. Without Blockchain technology, an automated process-based collaboration would be far more complex and restricted. Each participant may run his own workflow system on a local centralized data storage. After a collaboration is established, a malicious competitor is able to corrupt his local data storage and blame the other actors, if no trusted third party is included. The decentralized data storage in conjunction with the consensus mechanism on Blockchains prevents such a scenario. Albeit the advantages, our solution is not suitable for all choreography processes. Participating business partners have to agree on a globally accepted process model which is difficult, if sensitive data is included in the process or participants do not want to reveal organisation internal process flows. To the best of our knowledge, our approach is the only one, where the process logic is also on-chain, i.e. every participant can comprehend the steps and how the process is defined. In contrast, in [19] the process logic is defined off-chain directly within Solidity code for instance, which requires a kind of a trusted third party for initializing the process run and misses the point of Blockchain a bit.

Execution Costs. Transactions on the (public) Ethereum Blockchain are not free of charge. Miners are awarded for using their computational power to solve the proof-of-work. The amount of GAS refers to the complexity of the computations. The GAS multiplied with the user-set GAS price (a higher price leads to a faster validation) then indicates the price to pay. We propose a very different approach compared to former work, wherefore we have to investigate the execution costs for our implementation. The result of our benchmarks is depicted

Table 1. Time and GAS consumption for different number of tasks on our test network

Number of Tasks:	100	1000	5000	8000
Duration (function addTask):	14.36	148.98	742.5	1237.6
Average:	0.1463	0.1490	0.1485	0.1547
GAS (10^6):	12.7	1270	6350	10160
Duration (function setTaskOnCompleted)	11.38	114	482.6	802.5
Average:	0.1138	0.1140	0.0965	0.1003
GAS (10^6):	2.8	280	1400	2240

in Table 1. The (time) measurements bear on our local network solely, as we suggest the usage of a private/Consortium Blockchain anyhow for several reasons (see below). As Vitalik Buterin stated, the costs on a Consortium Blockchain can be significantly lower as just a few nodes must verify the transactions, instead of a world-wide network [4]. Then again, a Consortium Blockchain weakens the tamper-proofness a little bit, as the voting power in the system is concentrated on selected nodes. A sophisticated discussion on public vs. private/Consortium Blockchains related to BPM is still missing in research. Table 1 shows that even for oversized process executions with 8000 tasks, the architecture scales very well and the average duration for adding and completing a task respectively remains constant. The overall GAS consumption rises on a linear basis with

the amount of tasks and is $127,000$ for the transaction of adding one task to the Smart Contract and $28,006$ for the function call of completing one task. The initial deployment of the Smart Contract requires $1,265,261$ GAS. In [19], the execution of an Incident Management process containing 9 tasks costed on average 0.0347 Ether (ETH). The execution costs of a process with 9 tasks using our solution are the following: $1,265,261 + 9 \cdot 127,000 + 9 \cdot 28,009 = 2,660,342$. Based on the GAS price set, i.e. how fast the transactions are validated, the overall costs range from 0.00532 ETH and 0.0532 ETH which is on a similar level compared to the values of [19]. However, as they generate one function for each task and gate, and further on each function holds three different arrays for *PreviousElements*, *NextElements* and *NextJoins*, we believe that for more complex process models our approach with key-value stores scales much better.

Privacy and Security. Distributing all process related data affects the privacy of the data. Other publications address this issue by keeping the sensible data off-chain, or encrypt the data and use hash values to establish data integrity at least [19]. However, our architecture is designed to store as much data as possible on-chain (e.g. just extend the `struct Task`). This will further raise traceability, transparency and to make Blockchain technology attractive to advanced disciplines of Business Process Management like Process Mining etc. The usage of a Consortium Blockchain helps to address also this issue. All participants are known in this isolated network, thus the data sensibility must only be evaluated against the known business collaboration partners. On the other hand, security is also enhanced. Referring to the *DAO attack*[3] or the theoretical *51% attack*[4] in the Bitcoin network, Consortium Blockchains are not affected from these due to the isolation from the public network.

7 Conclusion

The usage of Blockchain technology empowers business partners to facilitate the collaboration within choreography processes. Having a Consortium Blockchain in action reduces the costs as the proof-of-work mechanism to reach consensus is spread over a preselected number of nodes. Hence, our solution is not forced to limit the number of transactions or the amount of data stored on the Blockchain. Thus, our proposal can be used in future research to include more concepts of BPMN for once and second, to include data attributes to support data-aware process execution. Further research has to keep up with latest innovations in the Blockchain universe. The technology is still immature and new ideas are discussed permanent. For instance, the consensus mechanism on the Ethereum Blockchain is planned to be changed from proof-of-work to proof-of-stake [2]. For the future we plan to support the mentioned data-aware processes and evaluate the integration of IoT devices in our process system on the Blockchain. We also want to develop further tool support like process-aware user-interfaces for wallets and the integration of our implementation into workflow management systems.

[3] https://www.coindesk.com/understanding-dao-hack-journalists/
[4] https://www.coindesk.com/bitcoin-miners-ditch-ghash-io-pool-51-attack/

References

1. Antonopoulos, A.M.: Mastering Bitcoin: Programming the Open Blockchain. O'Reilly Media, Inc. (2017)
2. Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. CoRR (2014), http://arxiv.org/abs/1406.5694
3. Buterin, V.: A next-generation smart contract and decentralized application platform. Tech. rep. (2014), https://github.com/ethereum/wiki/wiki/White-Paper
4. Buterin, V.: On public and private blockchains (2015), https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/
5. García-Bañuelos, L., Ponomarev, A., Dumas, M., Weber, I.: Optimized execution of business processes on blockchain (2017), https://doi.org/10.1007/978-3-319-65000-5_8
6. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs (2010), https://doi.org/10.4204/EPTCS.69.5
7. Huh, S.P., Cho, S., Kim, S.: Managing iot devices using blockchain platform. 2017 19th International Conference on Advanced Communication Technology (ICACT) (2017)
8. Hull, R., et al.: Towards a shared ledger business collaboration language based on data-aware processes. In: ICSOC (2016)
9. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I.: Caterpillar: A blockchain-based business process management system (2017)
10. Madsen, M.F., Gaub, M., Hgnason, T., Kirkbro, M.E., Slaats, T., Debois, S.: Collaboration among adversaries: Distributed workflow execution on a blockchain. Symposium on Foundations and Applications of Blockchain (2018)
11. Mendling, J., et al.: Blockchains for business process management - challenges and opportunities. CoRR (2017), http://arxiv.org/abs/1704.03610
12. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system, http://bitcoin.org/bitcoin.pdf (2008)
13. Neubauer, D.M., Goebel, A.: Blockchain for off-chain smart contracts in an sap environment (2018)
14. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference (2014), http://dl.acm.org/citation.cfm?id=2643634.2643666
15. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime verification for business processes utilizing the bitcoin blockchain. CoRR (2017), http://arxiv.org/abs/1706.04404
16. Rohr, J.: Blockchain for disaster relief: Creating trust where it matters most (2017), https://news.sap.com/blockchain-disaster-relief/
17. Szabo, N.: Formalizing and securing relationships on public networks. First Monday (1997), http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/548
18. Wang, W., et al.: A survey on consensus mechanisms and mining management in blockchain networks (May 2018)
19. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: Business Process Management - 14th International Conference, Proceedings (2016), https://doi.org/10.1007/978-3-319-45348-4_19