A Subscription Service for Automated Communication and Fair Cost Distribution in Collaborative Blockchain-based Business Processes

Moritz Schindelmann[1], Philipp Klinger[1], Freimut Bodendorf[1]

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Lehrstuhl für Wirtschaftsinformatik, insb. im Dienstleistungsbereich, Nürnberg, Germany
{moritz.schindelmann, philipp.klinger, freimut.bodendorf}@fau.de

Abstract. Blockchain capabilities like Ethereum Smart Contracts offer great opportunities to manage cross-organizational business processes due to their trustless and tamperproof nature. However, communication in such processes poses a major issue since there is no direct option for one participating organization to inform other collaborators about their individual progress in their impersonation as a Smart Contract. As that knowledge is vital to execute a cross-organizational process, we design a Smart Contract architecture in which participants express their progress through Blockchain events. Furthermore, we implement a prototype that subscribes to the relevant events of one or more participants and reacts to their occurrence by triggering the subsequent step(s) of the process. Evaluation of the prototype and architecture shows that this does not only avoid unnecessary latency in process communication but also results in a fair cost distribution as each participant is only charged for the expenses of its individual actions.

Keywords: Blockchain, Ethereum, Business Process Management, Cross-Organizational Collaboration.

1 Introduction

While Business Process Management promises great potential for cross-organizational processes, the use of such management systems is heavily restrained by the lack of mutual trust between process participants [1]. Blockchain-based solutions help to overcome that issue by building execution engines that enable running business processes in a trustless and tamperproof manner. The business logic of a process is thereby modeled through Smart Contracts stored on the Blockchain [2]. The conversion from a process model in BPMN to Smart Contracts should be abstracted from the user [3]. The basic idea is to run a translation algorithm during design time that parses and transforms BPMN into Smart Contract code [4]. Comparable solutions use choreography diagrams [5] or orchestration diagrams [4] as a starting point while modeling. Contrary, we choose to focus on collaboration models that require no further conversion before translation to Smart Contracts. As a

15th International Conference on Wirtschaftsinformatik,
March 08-11, 2020, Potsdam, Germany

https://doi.org/10.30844/wi_2020_r13-schindelmann

consequence, processes are easier to integrate into existing process-aware information systems. Also, participants may decide which process segments to execute using the Ethereum Blockchain. Per participant, that should be executed on-chain, one Smart Contract is created to respect the organizational boundaries in a collaborative process [4]. When executing such a process, the participants must communicate with each other. Contracts need to be invoked by other contracts (that may be traced back to a user-signed transaction in the first place) or directly user-signed transactions [6]. Good contract design is key to ensure cost-fairness amongst participants when executing the contracts. No participant should have to pay for actions of others. The effect of unnecessary expenses for a subset of participants should not be underestimated because fair cost distribution is vital in a collaboration of distinct organizations. If costs are unfairly distributed and incentives for execution do not align, companies are likely to avoid Blockchain-based solutions as additional and unfair costs would outweigh the Blockchain's benefits.

The goal of this paper is to answer the following research question: *How can we automate process communication whilst maintaining a fair cost distribution in Blockchain-based business process execution?*

We follow the Design Science Research (DSR) approach in order to answer the proposed research question. DSR is a research design distinguishing five phases, namely presentation of a problem, conception of its solution, implementation of a software artifact, its evaluation, and a conclusion. Every step of the design can yield new or more precise information and therefore lead to a new research iteration [7]. The solution put forth in this paper uses Smart Contract Events as a focal point. On Ethereum, we use a Smart Contract architecture that utilizes the events to log completed process actions of the participants. Off-chain, we implement a Subscription Service (SS) prototype that can dynamically subscribe to an arbitrary number of participants [and their emitted events]. If one of the subscribed participants emits an event, the service will analyze its content and trigger the subsequent step(s) of the business process. The service is evaluated by simulating an exemplary business process, tracking the metrics process latency and cost distribution, and comparing the results to those of an alternative approach for automating the process communication.

The remainder of this paper is structured as follows. In section 2, we provide the fundamental information about Blockchains and business processes on which the concepts of section 3 are based on. In section 4, we present the Subscription Service prototype. After evaluating the SS in section 5, we contrast our research question and the evaluation results of our prototype in section 6 and draw a conclusion.

2 Background

2.1 Blockchain and Smart Contracts

The Blockchain technology, first proposed by Nakamoto in the context of the Bitcoin cryptocurrency [8], is a distributed ledger in a peer to peer network. Peers are referred to as nodes and are equally responsible for storing the chain as a local copy and

https://doi.org/10.30844/wi_2020_r13-schindelmann

participating in the consensus mechanism to create new blocks, consequently achieving decentralization. Blocks are the data structure of the technology storing transactions and holding meta-data that links a block to the prior chain cryptographically with the predecessor's hash value. As the hash of every block is computed using information not only about its stored transactions but also about the previous blocks and the overall chain (e.g. its length), a manipulation of single transactions or entire blocks would lead to an incorrect hash making the error detectable, thus the data structure is described as immutable and tamperproof [9].

Ethereum allows storing executable programs, called contracts, initially written in Solidity. Contracts are stored in the form of bytecode, which is interpreted by the Ethereum Virtual Machine [10]. Interaction with deployed Smart Contracts requires their application binary interface (ABI) to encode the invocation from machine- into human-readable code and vice versa [11].

Contracts can either be invoked by call or transaction. While calls are limited to read operations, transactions execute state changes on-chain. Both transactions and calls can be initiated by a Smart Contract or an Externally Owned Account (EOA), but an initiating transaction must always be started by the latter. Ethereum also offers Smart Contract Events to log arbitrary data when performing transactions that might either provide extra information or simply improve traceability of one or more operations. Event arguments and their emitting Smart Contract addresses are stored in the transaction log which can be accessed by applications and interfaces [10].

2.2 Business Processes

A business process is a concatenation of coordinated activities that aim to realize a business goal [12]. No matter if cross-organizational or company internal, the better a process is structured and executed, the better is the resulting quality of its performance. Business process management systems (BPMS) use graphical elements to model processes and allow analysis, execution, monitoring, and evaluation with the use of their models [13]. There has been a recent uptake in Blockchain-based process execution endeavors based on Business Process Model and Notation (BPMN) diagrams [2, 4, 14].

2.3 Related Work

Management of cross-organizational business processes is often characterized by the lack of mutual trust and Blockchain-based solutions can be used to overcome that barrier [2]. Approaches range from process verification with the Bitcoin Blockchain [15] to monitoring and execution of processes based on Ethereum. Latest efforts either use standard BPMN notation [3], extend the BPMN standard [5, 16, 17], use other declarative workflow models [18], or rely on new modeling languages [19, 20].

The most widespread concept is based on Ethereum and works with (some form of) BPMN. In order to mimic the process flow, the business logic of a process model is converted to one or more Smart Contracts. Authors of [4] propose an important cornerstone, as they present an algorithm to automatically translate a choreography

https://doi.org/10.30844/wi_2020_r13-schindelmann

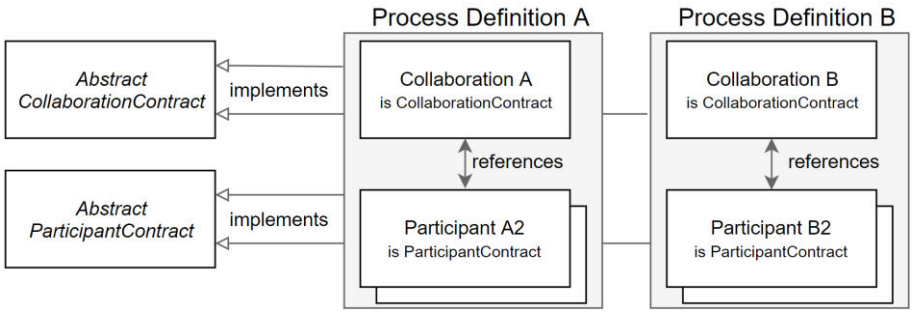
process model into Smart Contracts. Further works either also rely on choreographies [4, 5, 21], use single pools [22, 23], or translate the process models into state charts prior to Smart Contract conversion [24]. [4, 14, 24] are similar to our approach as Ethereum Events are used to log process states. However, none of the solutions work with Smart Contract Events to enable an optional service to automate process handovers in collaborations. A key benefit of our approach compared to previous works is, that no more conversion of the BPMN diagram is required. Furthermore, a process must not be performed entirely on-chain, thus integration of scenarios wherein only some participants rely on Blockchain-based execution is simplified.

Apart from related work in the context of process execution, the Subscription Service can also be associated with oracle services. The common purpose of oracles is to bring off-chain data onto the Blockchain to enable processing of real-world information [25]. Our service is different from a common oracle as the transferred information originates on the Blockchain.

3 Blockchain-based process execution with the Subscription Service

3.1 Smart Contract architecture for process models

The design of our Smart Contract architecture [26] distinguishes static build time and dynamic runtime contracts that either represent a collaboration or a participant. The static build time contracts are abstract classes providing basic data structures, functions, and events that are required for any given process. For every specific business process, a new set of contracts, containing one collaboration contract and one participant contract for every process collaborator, is implemented that fills the abstracts with the business logic of the individual process. For collaboration contracts, that means populating the data structures with the addresses of the participant contracts and the used instances. In contrast, participant contracts extend the structures and functions of the implemented interface with functions representing different steps a collaborator can execute. During the translation of the BPMN diagram into Smart Contracts, execution steps are equipped with a logging functionality using Ethereum Events.



https://doi.org/10.30844/wi_2020_r13-schindelmann

Figure 1. Simplified Smart Contract architecture based on [26]

Due to space limitations of this paper, we want to focus on explaining how our architecture accomplishes "*compliance by design*" [23], which prevents the Smart Contracts from performing actions that would be contrary to the logic of the modeled business process. Compliance is vital for our execution engine because the Subscription Service is an off-chain application and therefore possibly vulnerable to manipulation [25]. When an event is emitted by a subscribed participant contract, the SS initially checks that the event was sent by the expected participant, concerns a relevant process step, and has a valid process instance. If all that information is correct, the Subscription Service starts a transaction to call a function that represents the subsequent process step of another participant, thereby forwarding the event's information. To allow Smart Contracts to assure the function execution is conform to the business process, collaborations and their according participants are bidirectionally linked to each other. Furthermore, the collaboration contract holds two structures (among others), one for managing valid active process instances and another for keeping track of the progress of each participant in each process instance. These data structures allow a participant contract to assure three premises before executing a function. First, the participant reviews if the process instance, contained in the transaction, is valid. Second, the participant assures that it is in the correct state to execute the desired step, meaning that he has completed the prior process step but not yet started any later ones. Third, the participant confirms that the suggested event has actually taken place by reviewing the progress of the collaborator and examining that he has completed the logged process step in the transmitted process instance.

3.2 Standardized process fragmentation through wait states

Individual process parts must be separated into process steps to achieve the capability to programmatically define how process participants collaborate. For this purpose, wait states are a convenient concept to segment processes comprehensibly and consistently. A wait state describes a process milestone when a participant pauses its activities until it receives an external signal. That signal can be a user task that needs to be performed, an incoming event, or an incoming message [27].

https://doi.org/10.30844/wi_2020_r13-schindelmann

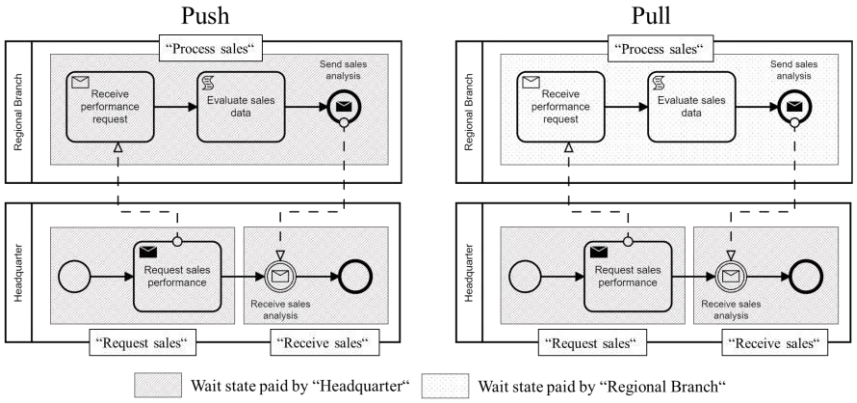


Figure 2. Wait states in an example process

As illustrated in Fig. 2, the application of the wait state concept onto a BPMN diagram results in at least one wait state for every participant as the interactions between the collaborators through message tasks or events always imply the border of a wait state. There are two basic approaches for automating the communication between participants, which we shall further refer to as "*pushing*" and "*pulling*". *Pushing* a process means actively starting the actions of another collaborator. In the context of our Blockchain-based system, that results in additional costs for the pushing participant as he is responsible for the expenses of the started activities. If the pushed participant's actions are all in a single wait state like the Regional Branch's in Fig. 2, that leads to an extreme cost divergence as the Headquarter has to pay for all of the Regional Branch's activities. In contrast, *pulling* the messages creates a reasonable share of the expenses. However, if one was trying to implement this concept solely with Smart Contracts, it would require contracts to constantly check the states of each other. As such cycling Smart Contracts cannot be implemented due to Ethereum's transaction model, an external application must take over this role instead, which is achieved by the Subscription Service in our case.

3.3 Storage of process data used by the Subscription Service

In order to allow the Subscription Service to understand how the collaborators and their wait states are connected in the business process, meta-data is needed. This meta-data comprises process- respectively message-flow information, as well as on-chain information like contract addresses, function signatures, and event signatures. Since data storage is expensive on Ethereum, we choose to store this meta-data in a local database for prototyping. This is a feasible approach, since the SS runs as a local (off-chain) component, too. The key information must be stored as follows. For each participant of every business process, every wait state must be stored that is subsequent to a message from a collaborator. It must be known what function signature on which contract address is represented by that wait state. Also, it must be comprehensible which event signature on which Smart Contract represents the

https://doi.org/10.30844/wi_2020_r13-schindelmann

message of the collaborator that the wait state's execution is dependent on and which content the emitted event is required to contain. All this information is directly extractable from the given BPMN diagram.

4 Implementation

4.1 Architectural Overview

Once collaborators have mutually agreed on a certain BPMN process, the diagram is consumed by the *BPMN-XML-Parser* and the *Transpiler*. The latter translates the process logic into Smart Contract code. It caches the contracts' bytecode, ABIs, and stores the deployment address in the participants' local artifact repositories. The *BPMN-XML-Parser* detects and extracts dependencies between the collaborators that are also stored in the participants' databases for further use by the Subscription Service.
Every participant disposes a local Subscription Service that it can use for any given collaboration stored in the environment in Fig. 3.

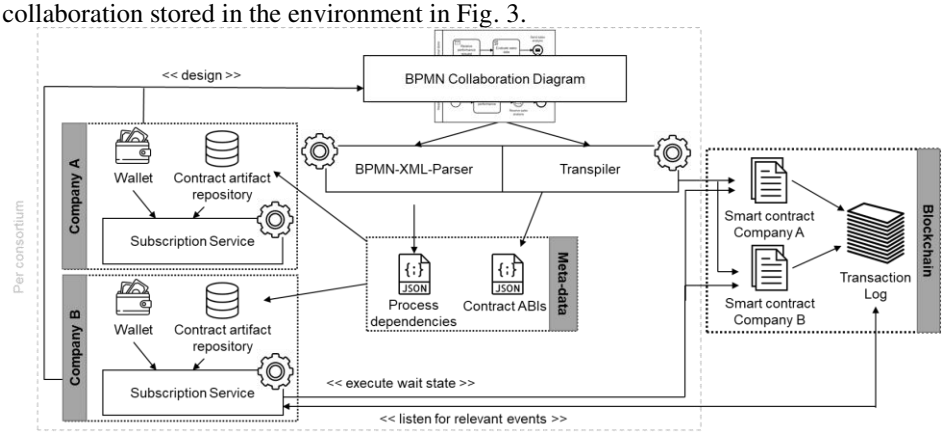


Figure 3. System architecture based on [26]

4.2 Core functionality of the Subscription Service

Our service uses Ethereum as an event store and relies on a Publish-Subscribe pattern [28] which distinguishes two roles. The publisher, a Smart Contract implementing the business logic, expresses information by emitting meaningful events to the Ethereum event log. A subscriber, which can be seen as our SS, is interested in certain content and therefore subscribes to specific publishers to consume the data of their emitted events. The Subscription Service uses this pattern to model process communication by receiving events and triggering subsequent process steps. The great advantages of this approach are the decentralization, immutability, longevity, and traceability of the store and its events as the data is protected by the Ethereum Blockchain.

https://doi.org/10.30844/wi_2020_r13-schindelmann

```
node SubscriptionService
    --C_COLLAB=0xe96E26c68CFCe0A2d4Ef2301BdB30c9B7884C2A2
    --C_PARTS="Participant_1ixny08,Participant_1v7ttq8"
    --N_HOST="localhost"
    --N_PORT=8545
    --PAY_ADD=0x06Aa5817208B97b4bB02C39B1928b9AB5c4aA859
```

Figure 4. Exemplary command for starting the Subscription Service

1. C_COLLAB: The first flag of the command expects the collaboration contract's deployment address. Every SS instance is only responsible for one collaboration, although multiple SS instances may run simultaneously.
2. C_PARTS: Participants, whose communication shall be automated by the SS instance, are identified by their respective ID attribute in the BPMN diagram. At least one participant is required, there is no upper limit.
3. N_HOST & N_PORT: Ethereum node endpoint connectivity information.
4. PAY_ADD: EOA used for transaction expenses by the SS. Accepts only one paying address argument per SS instance.

Initialization. When starting the Subscription Service, a connection to the given endpoint is established. Then, internal data structures for the interaction with the Smart Contracts of the collaboration and participants are initialized. The SS queries the relevant given participants' events from the local artifact repository. If no dependencies exist, the SS terminates. Otherwise, the service subscribes to events relevant for communication and waits for their occurrence.

Event occurrence. When a participant contract emits an event to communicate that a certain wait state is completed, the subscribed SS extracts its content. The service requires information about which process the participant belongs to (contract address of the collaboration), which participant logged the event (contract address of the participant), which process instance it concerns (ID of the process instance) and finally which wait state's completion the event communicates (unique label of the wait state). Based on this information, the service initially reviews if any relevant participant is depending on the logged wait state. In that case, the SS performs a quality check to confirm that the wait state was executed by the expected participant, concerns a valid and active process instance, and is connected to the fitting collaboration. Afterwards, the SS queries the database to learn which participant's wait state is supposed to be executed, and which function signature represents that state. When all data is assembled, the Subscription Service adds the information to the execution queue.

Execution queue. The queue is a numerically indexed data field that is polled regularly. For each observed item, the service checks if the belonging participant is ready for the execution of the queued wait state. Since states are represented through numerical indexes on the collaboration contract, the Subscription Service can determine if the current wait state of the participant is prior, equal, or subsequent to the state that shall be executed (desired wait state). Depending on the result of the comparison, the service handles the queue item in one of the following three ways.

1. Current Wait State < Desired Wait State: Keep item in the queue.
2. Current Wait State = Desired Wait State: Execute the wait state.

https://doi.org/10.30844/wi_2020_r13-schindelmann

3. Current Wait State > Desired Wait State: Remove item from the queue.

5 Evaluation and Discussion

In order to evaluate the latest iteration of our Subscription Service prototype, we simulate the execution of a business process, in which the participants want to use the BPMS solely for process verification, with a test script. Thereby, the participants are impersonated by Smart Contracts. Their actions (in form of functions) mimic the activities by changing the active wait state.

As shown in Fig. 5, the SS test distinguishes regular wait states and states that are executed by a Test Trigger. The script progresses through the evaluation process by successively trying to execute the triggers. The SS instances of the participants are expected to perform the subsequent wait states following a Test Trigger. Thereby, costs for the participants and the runtime of the overall process are tracked for each test run. To put the acquired results into context, the same Smart Contracts are used for a runtime optimized test script that uses the *push* concept. The difference between the two test scripts is that the *push* test does not only execute the Test Triggers but all wait states.

In terms of costs, our test results show that *pushing* the process communication leads to an unfair distribution, especially between the participants Headquarter and Repair Center. While the Repair Center does not pay for any of its actions, the expenses of the Headquarter are more than three times as high in comparison to the costs for its own activities. The test results of the *pull* approach show, contrary to the *push* concept, that the SS provides a fair distribution of expenses as each participant is responsible for its own actions only. Considering the latency and runtime of the process, the *push* concept is 15 seconds faster. The difference in Test Trigger 3 originates in the time needed by the SS to process the seven events of the wait states following the trigger.

https://doi.org/10.30844/wi_2020_r13-schindelmann

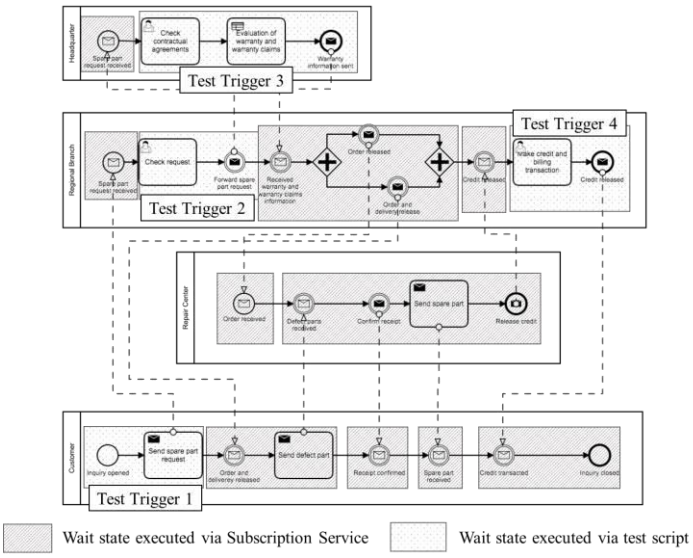


Figure 5. Example process used for evaluation of the SS

Table 1. Results of running the evaluation test scripts

Pull	**Costs [ETH]**					**Total [EUR]**	**Simulated Runtime [SEC]**
	Headquarter	Regional Branch	Repair Center	Customer	**Total**		
Trigger 1	0	0.0010320	0	0.0010302	**0.0020623**	**0.36**	**30**
Trigger 2	0.0010294	0.0007307	0	0	**0.0017601**	**0.31**	**30**
Trigger 3	0.0007937	0.0014610	0.0018244	0.0022551	**0.0063344**	**1.11**	**90**
Trigger 4	0	0.0007294	0	0.0012501	**0.0019795**	**0.35**	**30**
Total	**0.0018231**	**0.0039532**	**0.0018244**	**0.0045355**	**0.0121363**	**2.13**	**180**
Push	**Costs [ETH]**					**Total [EUR]**	**Simulated Runtime [SEC]**
	Headquarter	Regional Branch	Repair Center	Customer	**Total**		
Trigger 1	0	0	0	0.0020623	**0.0020623**	**0.36**	**30**
Trigger 2	0	0.0017601	0	0	**0.0017601**	**0.31**	**30**
Trigger 3	0.0063344	0	0	0	**0.0063344**	**1.11**	**75**
Trigger 4	0	0.0019795	0	0	**0.0019795**	**0.35**	**30**
Total	**0.0063344**	**0.0037396**	**0**	**0.0020623**	**0.0121363**	**2.13**	**165**

Tested with 20GWei Gas, 15 seconds block time, and price assumed to be at 175.43€ per Ether.

To put these findings into perspective, we must consider the major difference between the compared approaches. *Pushing* interactions between participants is time-efficient but also hardly applicable in public management systems due to its inability to fairly distribute costs between participants. Contrary, the *pull* concept is a reasonable approach that focusses on that exact problem to enable its usage in realistic use cases. Therefore, it must be stated that our Subscription Service is outperformed (in terms of process latency) by a runtime optimized approach that is not suitable for a real-world execution engine. Test results show that the SS should be considered time efficient and time-saving because the service automates process communication

https://doi.org/10.30844/wi_2020_r13-schindelmann

whilst retaining fair cost distribution in marginally slower runtime than the *push* concept. The difference in latency between the evaluated concepts has minor significance in a real process since the greatest reason for process latency is waiting time [29] which the SS reduces massively.

6 Conclusion

We find that collaborative business processes are rarely managed using business process management systems because such processes require a lacking trust between the participants to mutually design and execute the process. Blockchain-based management systems can solve this issue due to the immutable, trustless and tamperproof nature of the Blockchain technology. Yet, reaching fair cost distribution in such a setting is an unsolved problem. For this reason, we implement a Subscription Service for automated communication and fair cost distribution in collaboratively executed business processes on the Ethereum Blockchain. The Subscription Service subscribes itself to relevant participants of the collaboration and reacts upon emitted Ethereum Event logs accordingly. If a trigger event is noticed, the Subscription Service executes follow-up actions signing transactions using a provided secret key. Consequently, costs may be fairly attributed to the responsible participants. Another positive effect is the reduction of organizational waiting time, since follow-up actions may be triggered automatically, instead of manually checking the executability. In order to evaluate how our service performs in the context of automation and fair cost distribution, we compare the prototype to a *pushing* concept where Smart Contracts collaborate through starting transactions on another contract in order to express process communication. The results show that the SS enables a fair distribution of expenses between the participants of the process while providing the potential to reduce a large amount of organizational waiting time in processes. The limitations and starting points for further research especially concern the runtime environment of the Subscription Service. As of now, the *BPMN-XML-Parser* is limited to simple message events and tasks and does not support more sophisticated BPMN diagrams that contain complex gateways with timer events, subprocesses, or boundary events for example. Furthermore, our execution engine disregards the management of private keys used by the participants to sign transactions. Full BPMN support and key management would be valuable next steps to enhance the usability of our execution engine.

References

1. Breu, R., Dustdar, S., Eder, J., Huemer, C., Kappel, G., Julius, K., Langer, P.: Towards Living Inter-Organizational Processes. (2013). https://doi.org/10.1109/CBI.2013.59.
2. Mendling, J., Dustdar, S., Gal, A., García-Bañuelos, L., Governatori, G., Hull, R., Rosa, M. La, Leopold, H., Leymann, F., Recker, J., Reichert, M., Weber, I., Reijers, H.A., Rinderle-Ma, S., Solti, A., Rosemann, M., Schulte, S., Singh, M.P., Slaats, T., Staples, M., Weber, B., Weidlich, M., Aalst, W. Van Der, Weske, M., Xu, X., Zhu, L.,

https://doi.org/10.30844/wi_2020_r13-schindelmann

Brocke, J. vom, Cabanillas, C., Daniel, F., Debois, S., Ciccio, C. Di, Dumas, M.: Blockchains for Business Process Management - Challenges and Opportunities. ACM Trans. Manag. Inf. Syst. 9, 1–16 (2018). https://doi.org/10.1145/3183367.

3. Di Ciccio, C., Cecconi, A., Dumas, M., García-Bañuelos, L., López-Pintado, O., Lu, Q., Mendling, J., Ponomarev, A., Binh Tran, A., Weber, I.: Blockchain Support for Collaborative Business Processes. Inform. Spektrum. 42, 182–190 (2019). https://doi.org/10.1007/s00287-019-01178-x.
4. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted Business Process Monitoring and Execution Using Blockchain. In: International Conference on Business Process Management (2016). https://doi.org/10.1007/978-3-319-45348-4_19.
5. Ladleif, J., Weske, M., Weber, I.: Modeling and Enforcing Blockchain-Based Choreographies. In: International Conference on Business Process Management. pp. 69–85 (2019). https://doi.org/10.1007/978-3-030-26619-6_7.
6. Wood, G.: Ethereum: A Secure Decentralised Generalised Transaction Ledger, (2014).
7. Vaishnavi, V., Kuechler, W., Petter, S.: Design Science Research in Information Systems. Advances in Theory and Practice. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29863-9.
8. Nakamoto, S.: Bitcoin : A Peer-to-Peer Electronic Cash System.
9. Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S.: Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press, Princeton (2016).
10. Wöhrer, M., Zdun, U.: Smart contracts: security patterns in the ethereum ecosystem and solidity. In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE). pp. 2–8. IEEE (2018). https://doi.org/10.1109/IWBOSE.2018.8327565.
11. Mühlberger, R., Bachhofner, S., Di Cicco, C., Garcia-Banuelos, L., Lopez-Pintado, O.: Extracting Event Logs for Process Mining from Data Stored on the Blockchain. In: International Conference on Business Process Management (Workshop Proceedings) (2019).
12. Weske, M.: Business Process Management. Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28616-2.
13. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, Berlin, Heidelberg (2018).
14. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I.: Caterpillar: A Blockchain-based Business Process Management System. In: International Conference on Business Process Management (Demo Track) (2017).
15. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime verification for business processes utilizing the Bitcoin blockchain. Futur. Gener. Comput. Syst. (2017). https://doi.org/10.1016/j.future.2017.08.024.
16. Falazi, G., Hahn, M., Breitenbücher, U., Leymann, F.: Modeling and execution of blockchain-aware business processes. Software-Intensive Cyber-Physical Syst. 34, 105–116 (2019). https://doi.org/10.1007/s00450-019-00399-5.
17. Tran, A.B., Lu, Q., Weber, I.: Lorikeet : A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management. In:

International Conference on Business Process Management (Workshop Proceedings) (2018).
18. Madsen, M.F., Gaub, M., Høgnason, T., Kirkbro, M.E., Slaats, T., Debois, S.: Collaboration among Adversaries : Distributed Workflow Execution on a Blockchain. In: Symposium on Foundations and Applications of Blockchain (FAB '18) (2018).
19. Hull, R., Batra, V.S., Chee, Y.-M., Deutsch, A., Heath, F., Vianu, V.: Towards a Shared Ledger Business Collaboration Language Based on Data-Aware Processes. In: ICSOC 2016. pp. 202–218 (2016). https://doi.org/10.1007/978-3-319-46295-0.
20. López-Pintado, O., Dumas, M., García-Bañuelos, L., Weber, I.: Dynamic Role Binding in Blockchain-Based Collaborative Business Processes. In: International Conference on Advanced Information Systems Engineering (2019).
21. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime Verification for Business Processes Utilizing the Bitcoin Blockchain Runtime Verification for Business Processes Utilizing the Bitcoin Blockchain. (2017). https://doi.org/10.1016/j.future.2017.08.024.
22. García-bañuelos, L., Ponomarev, A., Dumas, M., García-bañuelos, L., Ponomarev, A.: Optimized Execution of Business Processes on Blockchain Optimized Execution of Business Processes on Blockchain. In: International Conference on Business Process Management (2017). https://doi.org/10.1007/978-3-319-65000-5.
23. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., Ponomarev, A.: Caterpillar: A business process execution engine on the Ethereum blockchain. In: International Conference on Business Process Management (Workshop Proceedings) (2019). https://doi.org/10.1002/spe.2702.
24. Nakamura, H., Miyamoto, K., Kudo, M.: Inter-organizational Business Processes Managed by Blockchain. In: International Conference on Web Information Systems Engineering (2018). https://doi.org/10.1007/978-3-030-02922-7_1.
25. Teutsch, J.: On decentralized oracles for data availability, https://people.cs.uchicago.edu/~teutsch/papers/decentralized_oracles.pdf, last accessed 2019/10/10.
26. Klinger, P., Bodendorf, F.: Blockchain-based Cross-Organizational Execution Framework for Dynamic Integration of Process Collaborations. In: 15th International Business Informatics Congress (WI2020) (2020).
27. Camunda: Transactions in Processes, https://docs.camunda.org/manual/7.8/user-guide/process-engine/transactions-in-processes/, last accessed 2019/10/13.
28. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.-M.: The Many Faces of Publish/Subscribe. ACM Comput. Surv. 35, 114–131 (2003). https://doi.org/10.1145/857076.857078.
29. Belkin, V.: Multikriterielles Controlling von Geschäftsprozessen: Prozessverbesserung mit Hilfe der dynamischen Simulation. Eul, Lohmar (2011).