Extracting Event Logs for Process Mining from Data Stored on the Blockchain

Roman Mühlberger, Stefan Bachhofner, Claudio Di Ciccio, Luciano García-Bañuelos and Orenys López-Pintado

EasyChair preprints are intended for rapid dissemination of research results and are integrated with the rest of EasyChair.

September 4, 2019
Extracting Event Logs for Process Mining
from Data Stored on the Blockchain

Roman Mühberger¹, Stefan Bachhofner¹,
Claudio Di Ciccio¹, Luciano García-Bañuelos²,³, and Orlenys López-Pintado²

1 Vienna University of Economics and Business, Austria
   {roman.muehlberger}@s.wu.ac.at,
   {stefan.bachhofner,claudio.di.ciccio}@wu.ac.at
2 University of Tartu, Estonia
   orlenyslp@ut.ee
3 Tecnológico de Monterrey, Mexico
   luciano.garcia@tec.mx

Abstract. The integration of business process management with blockchains across organisational borders provides a means to establish transparency of execution and auditing capabilities. To enable process analytics, though, non-trivial extraction and transformation tasks are necessary on the raw data stored in the ledger. In this paper, we describe our approach to retrieve process data from an Ethereum blockchain ledger and subsequently convert those data into an event log formatted according to the IEEE Extensible Event Stream (XES) standard. We show a proof-of-concept software artefact and its application on a data set produced by the smart contracts of a process execution engine stored on the public Ethereum blockchain network.

Keywords: Ethereum · Process Discovery · Process Monitoring · Process Conformance

1 Introduction

Blockchain offers a new prospective environment for the execution of inter-organisational processes [24]. On top of a distributed execution environment, blockchains collate transactions onto a push-only stack (ledgers) within backward-linked blocks [25]. Participants are provided with an up-to-date version of the entire blockchain via broadcast after a new block has been added following a consensus algorithm. The utilisation of the blockchain in processes covering a number of participants prevents the necessity of a central orchestrator and provides trust among mutually untrusted parties [31]. Additionally, the decentralised database stores transactional information triggered by smart contracts containing the state and execution details of distinct cases [24]. Data can be retrieved that contains, inter alia, interacting process participants, (discrete) time and duration of the interaction, execution costs on the blockchain as well as the state details of a limited, recurring set of activities. This information enables process monitoring [7], process compliance [32], and process conformance checking [23,26] via process mining [29,30]. In addition, the information is also valuable for the analysis of the efficiency and effectiveness of process executions, i.e., for performance mining [15,11,19,4].

In order to make proper use of that information, manual effort is required to convert the data from individual blocks into an appropriate format, which causes a variety of challenges [27]. To mention but a few, the information retrieved from the blockchain is represented
in hexadecimal and numeric formats; timestamps are approximate as the finest granular unit of time in blockchains is at the level of blocks; the structure of data payloads attached to transactions is for the most part arbitrary. Process mining tools, on the other hand, present clear constraints on data types and their required representation. In order to localise and convert the data, a profound understanding of the data model represented on the blockchain is essential.

In this work-in-progress paper, we follow the Design Science research methodology to devise a framework that retrieves process data from blocks and transforms the process data into an event log, formatted according to the IEEE Extensible Event Stream (XES) standard. The software artefact is reflected in an implemented prototype, which is applied on a real-world case study. The XES output file is used as an input for existing tools in the ProM toolkit. Additionally, we discuss the challenges and opportunities pertaining to data extraction, transformation and interpretation of blockchain data to drive future work in this area. To the best of our knowledge, this work is the first one retrieving and transforming blockchain data into a standardised format that can be used in process mining tools. Consequently, our work is the first that enables process monitoring, process conformance and process compliance checking with process mining on blockchain-based process executions.

The remainder of this paper is structured as follows. In Section 2, the notions backing our research work are provided. Section 3 describes our approach. Section 4 describes the experiment conducted with a real-world case study. The paper concludes with Section 5 including implications for further research.

2 Background

A blockchain consists of a distributed ledger of transactions [25]. Blockchains operate on top of a peer-to-peer network, where each peer stores a local copy of the ledger’s transactional data. The transactions are collated in blocks, which are sequentialised as an append-only chain. Compact identification and storage of data, as well as the backward-linking of blocks in the chain, are based upon hashing algorithms such as the SHA-3 compliant algorithm KECCAK [6]. Appending new blocks in a trustworthy manner is enabled by a combination of consensus-making, cryptography, and market mechanisms [25]. A central authority is thus not necessary to that extent, making it advantageous for processes containing a set of untrusted and unknown participants [31]. More recent blockchain protocols such as Ethereum [33] enable programmability of the platform through the smart contracts. Ethereum distributions come endowed with dedicated smart-contract programming languages such as Solidity [10], an object-oriented, high-level programming language. Ethereum smart contracts are executed in a decentralised environment named Ethereum Virtual Machine (EVM). Their function signatures and public attributes are exposed through a human- and machine-readable interface called Application Binary Interface (ABI). The execution of smart contract operations is associated to a price expressed in terms of gas, exchanged at a variable rate with the Ether cryptocurrency. The invocation of smart contracts’ functions occurs via transactions that are stored on the ledger.

Blockchain for business process execution. Smart contracts allow for the codification of business process logic on the blockchain [12], as shown in the seminal work of Weber et al. [31] and later on with tools like Caterpillar [22] and Lorikeet [28]. As several modern Business Process Management Systems (BPMSs) do, Caterpillar and Lorikeet adopt a
Fig. 1: The choreography Business Process Model and Notation (BPMN) model of an incident management process [31].

Model-Driven Engineering (MDE) approach to let the process analysts provide graphical representations of the process and turn it into executable code enacting it. They take as input models that are specified in the BPMN language [15]. Figure 1, e.g., illustrates the BPMN choreography model of the incident management process introduced by Weber et al. [31] to evaluate their execution engine for inter-organisational processes.\(^4\) The process begins with an issue raised by a customer. The key account manager registers the problem description. Until a solution is not found, the ticket escalates to the 1st level support, then to the 2nd level support, and finally to the developer. After the solution is found, a feedback is given and the solution explained to the customer.

The workflow routing of blockchain-based process execution engines is performed by smart contracts generated by compilers that translate BPMN diagrams into smart contract code (e.g., Solidity). Once deployed on the blockchain, the contract instances are identified by the hexadecimal address of their account, which serve as a key for querying the state of the process instances.

The transactional storage of interactions among the actors operating on the blockchain and with smart contracts enables traceability of processes [13]. Our aim is to extract the process data from the ledger and turn it into a readily processable format for process mining tools, in order to enable analysis and auditing of processes run on the blockchain.

**Event logs for process mining.** Process mining makes use of the sequential data stored by BPMS and other process-aware information systems such as Enterprise Resource Plannings (ERPs) or Customer Relationship Managements (CRMs) in order to discover, analyse and enhance existing processes [1]. As such, process mining is a valuable means to conduct auditing and forensics on existing process data [18]. The standard data structure with which those data are stored is named *event log*, or log for short. An event log consists of a collection of traces, where every *trace* represents a process run (or process instance) and activity executions

\(^4\) Diagram in Fig. 1 courtesy of Ingo Weber.
Table 1: An event log excerpt.

<table>
<thead>
<tr>
<th>Case ID</th>
<th>Event ID</th>
<th>Activity name</th>
<th>Timestamp</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Customer has a problem</td>
<td>2016-03-22T14:06:22.000Z</td>
<td>Key Account Manager</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Get problem description</td>
<td>2016-03-22T14:06:35.000Z</td>
<td>Key Account Manager</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Ask 1st level support</td>
<td>2016-03-22T14:07:00.000Z</td>
<td>Key Account Manager</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>Ask 2nd level support</td>
<td>2016-03-22T14:08:30.000Z</td>
<td>1st Level Support</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>Provide feedback for 1st level support</td>
<td>2016-03-22T14:09:00.000Z</td>
<td>2nd Level Support</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>Provide feedback for account manager</td>
<td>2016-03-22T14:09:14.000Z</td>
<td>1st Level Support</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>Explain solution</td>
<td>2016-03-22T14:09:22.000Z</td>
<td>Key Account Manager</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Customer has a problem</td>
<td>2016-03-22T14:37:45.000Z</td>
<td>Key Account Manager</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>Get problem description</td>
<td>2016-03-22T14:38:00.000Z</td>
<td>Key Account Manager</td>
</tr>
<tr>
<td>32</td>
<td>250</td>
<td>Customer has a problem</td>
<td>2016-03-22T12:46:01.000Z</td>
<td>Key Account Manager</td>
</tr>
<tr>
<td>32</td>
<td>251</td>
<td>Get problem description</td>
<td>2016-03-22T12:46:21.000Z</td>
<td>Key Account Manager</td>
</tr>
<tr>
<td>32</td>
<td>252</td>
<td>Ask 1st level support</td>
<td>2016-03-22T12:47:16.000Z</td>
<td>Key Account Manager</td>
</tr>
<tr>
<td>32</td>
<td>253</td>
<td>Ask 2nd level support</td>
<td>2016-03-22T12:47:53.000Z</td>
<td>1st Level Support</td>
</tr>
<tr>
<td>32</td>
<td>254</td>
<td>Provide feedback for 1st level support</td>
<td>2016-03-22T12:48:00.000Z</td>
<td>2nd Level Support</td>
</tr>
<tr>
<td>32</td>
<td>255</td>
<td>Provide feedback for account manager</td>
<td>2016-03-22T12:48:40.000Z</td>
<td>1st Level Support</td>
</tr>
<tr>
<td>32</td>
<td>256</td>
<td>Explain solution</td>
<td>2016-03-22T12:49:17.000Z</td>
<td>Key Account Manager</td>
</tr>
</tbody>
</table>

Fig. 2: The meta model class diagram representation of the XES standard [17].

are represented by events (each mapped to an activity name). Event logs may contain additional information in the form of event attributes, such as the timestamp or the resource carrying out the related activity. Table 1 shows an excerpt of event log that may originate from the execution of the process in Fig. 1. Process analytics toolkits such as ProM [2] or Apromore [20] take as input event logs that are formatted according to the XML-based IEEE standard XES [17]. Figure 2 illustrates the original version of the standard. The top level element of XES is a log containing trace nodes (one per process instance). Traces incorporate event nodes. Data are stored in attributes, which are represented as key-value nodes. The XES standard defines attribute data types and semantics through an extension mechanism. Every extension has a name, a prefix, and a reference Uniform Resource Identifier (URI). For example, extension Concept, with prefix concept:, and URI http://www.xes-standard.org/concept.xesext, is used to name elements such as event (typically with the corresponding activity label) and traces. Other
Extracting Event Logs for Process Mining from Data Stored on the Blockchain

3 Approach

Figure 3 illustrates our approach for the extraction and transformation to XES of process data from the blockchain. We remark that we use only information that is stored on the blockchain.

We begin with the analysis of the smart contract implementing the business process. With this analysis, we aim at producing information to identify transactions that are associated with the execution of process tasks. To that end, we rely on the fact that the payload of an Ethereum transaction stores the function selector, i.e., the first four bytes in the SHA-3 hash of a string encoding the function signature. For instance, let us consider task Customer has a problem, which is implemented by a function with signature Customer_Has_a_Problem(). The selector for such function is 0xe73dcb, because the SHA-3 hash for Customer_Has_a_Problem() is 0xe73dcb348c11a7ab31ce1e20102e63c94e84ab393a78f187d1485c8a2c72cc. For convenience, we keep a hash-table to associate task names with their corresponding function selectors, as we use it to generate and enrich the event log.

Blockchain-based BPMSs such as Caterpillar and Lorikeet use the factory pattern to control the creation of process instances: a factory contract deploys, for each new process instance, a smart contract that enacts the business process. We observe that the address of the factory contract is included in the metadata of the transaction associated with the creation of a new process instance as the originator of such transaction, i.e., within the attribute Transaction.from. Therefore, if we assume that the address of the factory contract is provided, we can identify the address of each of the process instances created by the factory contract and later each one of the transactions associated with a process instance. We inspect the blockchain data model of blocks and transactions as illustrated in Fig. 4. We identify the hash as a unique identifier for every block and every transaction.
Fig. 4: The data model of blocks and transactions. We depicted the attributes used for the extraction of blockchain data.

The transactions included in a block can be obtained in the `Block.transactions` field of a block. The UNIX timestamp represented in the block can be used as a basis to derive the timestamps of the activities therein. The transactions are allocated to an enumeration denoting the order of the process executions and saved into a data structure. In order to provide completeness of the dataset, we decode the hash values and save both the function signatures and activity names in a human-readable format. In order to add the activity name, we compare the data from the transaction input field with the signature hash saved in the data structure, which is denoted as Function signature data store in Fig. 3. For consistency reasons, we use the transaction data from the data structure denoted as Instance transactions data store to add additional information such as the costs of the process instances in Ether or fiat currency like US dollars. Finally, the data extraction and processing is finalised and the result can be transformed into an event log that is compliant with the XES format.

To transform the instance transactions data store into a log that complies with XES standard, we map the attributes in the data store to XES event and trace attributes. Whenever suitable, we append XES extension declarations to our file to enrich events with additional information, such as cost or time, declared into the header of the XES document. For example, we use the mentioned transaction hash 0x656252f3ecee102d981520ca9e0ca0f7048bce99e4f6f8d89d358c6156 as the id node of the event. We use the function selector, i.e., the first four bytes of the `Transaction.input` field (0xefe73dcb) as a custom activity_id attribute. Timestamp 2016-03-22T13:44:22.000Z is mapped to the event with the key time:timestamp.

4 Case Study

In this section, we present a case study for our approach. We extract process data from a public blockchain and transform the process data into a XES log. The software project, including the Jupyter notebook for our case study, is openly available as a public GitLab repository.6

4.1 Extraction of an event log from the public Ethereum blockchain

To evaluate our approach, we analyse the transactions stored on the public Ethereum network by the execution engine described by Weber et al. [31]. Figure 1 illustrates the BPMN choreography model of the incident management process they implemented. The process is enacted by

---

Table 2: Mapping between event log elements and blockchain data fields in our case study.

<table>
<thead>
<tr>
<th>Event log attribute</th>
<th>Blockchain data field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case ID</td>
<td>Transaction.to</td>
</tr>
<tr>
<td>Activity ID</td>
<td>Transaction.input[2:10] (function selector)</td>
</tr>
<tr>
<td>Event ID</td>
<td>Transaction.hash</td>
</tr>
<tr>
<td>Activity label</td>
<td>Reverse-engineered from contract ABI and function selector in Transaction.input</td>
</tr>
<tr>
<td>Event timestamp</td>
<td>Block.timestamp (plus sorting by Transaction.index)</td>
</tr>
<tr>
<td>Event cost</td>
<td>Transaction.gas</td>
</tr>
<tr>
<td>Event resource</td>
<td>Transaction.from</td>
</tr>
</tbody>
</table>

Listing 1: Key-value representation of the transaction data denoting the execution of an activity.

```
{
    blockHash: 0x1eca7ae74de59dff4f553b055a0e8346b1f1587c7f6ca5a0b22da84f8782b,
    blockNumber: 1196772,
    from: 0x1387e74982055e3e1d235aad579350813b329b2b,
    gas: 1000000,
    gasPrice: 20000000000,
    hash: 0xe55252f3e1102698152ca5e9c5ac0f48bcae9e4f6fe6a9e356cd53d6d6156,
    input: 0xefe73dcb,
    nonce: 207,
    r: 0xf26831a097e1a3cf64364a07ff80fa816dd860461482921a81be747fb5ab7,
    s: 0xe6a4b1b999b0d04d155d0a4cd6e3c329ab5b1d0e9c0b0fccc8a5cfa9965f6d,
    to: 0x0e641313d8e18a7488c6e62eef77ea8e6e9a9f73f,
    transactionIndex: 3,
    v: 28,
    value: 0
}
```

The process instances implemented on the blockchain are initiated by a factory smart contract at address 0x09890f52cdd5d0743c7d13abe481e705a2706384. The factory contract deploys a new instance of process contract for every new run. To retrieve the process data, we access the Etherscan Ethereum blockchain explorer and analyse the list of transactions having the factory smart contract as the sender, i.e., such that the factory address occurs in their Transaction.from field. Those transactions amount to 32 in our case.7

Table 2 summarises the mapping between the event log elements and the blockchain data. The Transaction.to field of each of the transactions from the factory contract identifies a process instance, thus 32 process instances were run on the blockchain. Using again Etherscan, we retrieve the transactions directed to every such process instance contract. Listing 1 shows one of those transactions. To identify which of those correspond to an activity enactment, we operate as follows. Considering the set of activities of the process, we find the corresponding function names in the ABI of the called contract. Thereupon, we compute the KECCAK hash for each of the function signatures of interest and save the hexadecimal representation of its first four bytes, as shown in Table 3. We thus select the transactions denoting the activity enactment by selecting the ones that have those four bytes as the prefix of their Transaction.input field. The transaction of Listing 1, e.g., corresponds to activity `Customer has a problem`.

Finally, we turn the extracted transactions into a XES document. Every process instance contract corresponds to a trace. Therefore, we use its hash as the trace identifier (id). Every (activity-related) transaction towards that contract maps to an event therein, so its hash represents the id of the event. As an additional attribute we add XES-defined standard

---

7 https://etherscan.io/address/0xe09890f52cdd5d0743c7d13abe481e705a2706384
Table 3: The activity names, their associated smart contract’s function signatures, and the first four bytes of the KECCAK hash of those signatures, for the process in Fig. 1.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Function signature</th>
<th>Function selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer has a problem</td>
<td>Customer_Has_a_Problem()</td>
<td>0xefe73dcb</td>
</tr>
<tr>
<td>Get problem description</td>
<td>Get_problem_description(int32 x)</td>
<td>0x92ed10ef</td>
</tr>
<tr>
<td>Ask 1st level support</td>
<td>Ask_1st_level_support(int32 y)</td>
<td>0x2b06df7</td>
</tr>
<tr>
<td>Explain solution</td>
<td>Explain_solution()</td>
<td>0x80ef07f19</td>
</tr>
<tr>
<td>Ask 2nd level support</td>
<td>Ask_2nd_level_support()</td>
<td>0x63ad6b81</td>
</tr>
<tr>
<td>Provide feedback for account manager</td>
<td>Provide_feedback_for_account_manager()</td>
<td>0x5b866413</td>
</tr>
<tr>
<td>Ask developer</td>
<td>Ask_developer()</td>
<td>0x0cb7b8c</td>
</tr>
<tr>
<td>Provide feedback for 1st level support</td>
<td>Provide_feedback_for_1st_level_support()</td>
<td>0x3b26a0ea</td>
</tr>
<tr>
<td>Provide feedback for 2nd level support</td>
<td>Provide_feedback_for_2nd_level_support()</td>
<td>0x9ec3200a</td>
</tr>
</tbody>
</table>

Listing 2: XES event corresponding to the transaction of Listing 1.

```
<event>
  <string key="id" value="1ox656232f3ceee302d881520ca9e0ca0f7048be3099e4f4f0ed89e358c9bedc615e" />
  <string key="concept:name" value="Customer␣has␣a␣problem" />
  <string key="activity_id" value="e0xefe73dcb" />
  <string key="cost:currency" value="USD" />
  <date key="time:timestamp" value="2016-03-22T13:44:22.000Z" />
  <string key="lifecycle:transition" value="complete" />
  <float key="cost:total" value="0.1754522" />
  <int key="blockNo" value="1196772" />
  <float key="feeEth" value="0.0011393" />
  <float key="pricePerEth" value="10.96" />
  <string key="org:resource" value="0x1387e74982055e3e1d235aad579350813b329b2b" />
</event>
```

4.2 Analysis of the event log with ProM

We used the generated event log as an input for ProM to test its usage for process mining. A full investigation and analysis of the event log goes beyond the scope of this paper. This preliminary experiment shows the possibilities opened up by our approach, which extracts and transforms data stored on the blockchain to make it readily available for process mining. Figure 5 illustrates the event log imported in ProM and visualised through the Dotted Chart plugin. The id of traces is on the y axis and on the x axis we see the timestamp of events. Traces are sorted by the timestamp of their first event. The colour of dots corresponds to the name (activity label) of events. It can be noticed that the process instances were run

sequentially at close distance in time. Figure 6 depicts the output of the Inductive Visual Miner discovery plug-in using the standard set-up [21]. Considering the original model in Fig. 1, we can observe that all instances were such that the 2nd-level support resolved the issue as activities \textit{Ask developer} and \textit{Provide feedback for 2nd level support} were never executed.

Our observation is confirmed by the application of the conformance checking plug-in of Adriansyah et al. [3] on the log and on a Workflow net that simulates the behaviour of the original process [1], as illustrated in Fig. 7. As it can be noticed in the information panel on the left-hand side, the model is 100% fitting with the recorded traces. Indeed, the border of all activities (transitions, graphically depicted as boxes) are surrounded by a green line. The background colour of transitions indicates whether the corresponding tasks were executed
(blue) or not (white). Notice that, as expected, the two activities at the bottom of the diagram (Ask developer and Provide feedback for 2nd level support) did not occur. A full-fledged quantitative analysis of the process is beyond the scope of this paper. However, the presented outcome hints at the possibilities that the creation of event logs out of transaction data on the blockchain opens up.

4.3 Discussion and limitations

The described approach shows promising results. However, we acknowledge limitations that are inherently bound to the data analysis we conduct. For process mining, the information found on the blockchain is required to include at least unique identifiers for process instances, timestamps of executions as well as identifiable activity denotations. In our case study, we detect unique identifiers and activity names by considering a smart contract function call as the unit of execution, i.e., the event. However, we understand that smart contract functions may not match process activities. This calls for more advanced techniques to relate smart contract transactions to process task executions [5]. Also, we associate process instances to smart contract life-cycles. This assumption holds when every process run corresponds to one and only one contract. Blockchain BPMNs that adopt other architectural patterns than the factory one could make our assumption not valid any longer, thus requiring more sophisticated reference reconciliation mechanisms, or necessitating transaction payloads to bear a unique identifier for instances. Furthermore, we approximate event timestamps with the block time, although the level of granularity may not be sufficient and fully reliable. Information stemming from off-chain sources should be retrieved from certified sources such as the so-called oracles [34]. This opens up new challenges for future work, aimed at mixed on-chain/off-chain information retrieval approaches. In general, our work depends on the data model retrieved from the transactions. In fact, for different cases an adjustment to the corresponding data model may be necessary. In future research endeavours, algorithms from semantic technologies could be leveraged to circumvent manual adjustments [16,8]. Also, we observe that our solution stores the generated event logs off-chain. This could make their synchronisation and update with new process runs harder and potentially hinder run-time monitoring. To circumvent this issue, we envisage solutions that lie at the core blockchain architecture or leverage the support of additional data stores: an extension of blockchain protocol implementations with on-node storage of state and event information [9], or the adoption of hash-based links connecting transaction data with distributed file systems such as InterPlanetary File System (IPFS).9

5 Conclusion

In this paper, we described an approach to generate XES-compliant event logs out of process data stored on the blockchain. Our approach provides a blueprint to retrieve process data from the transactions ledger. As a proof-of-concept, we implemented a software prototype applying our approach on a case study, based on a blockchain-enabled process run on the public Ethereum network.

Our approach shows promising preliminary results, on the basis of which we envision a number of future research avenues. From a practical perspective, we are developing further

9 https://ipfs.io
software artefacts to apply our approach on other blockchain-based BPMSs such as Caterpillar [22]. Furthermore, we aim at investigating how to extract and process information stemming from other Distributed Ledger Technologies (DLTs) such as Hyperledger Fabric [14].

We argue that the main challenge in mining processes from the blockchain is the mapping between the process-specific data and their model and the concrete representations on the blockchain. Challenges arise if that mapping does not ensure traceability. Future research should thus be devoted to the creation of a modelling language describing how blockchain-based process interactions are mapped onto blockchain data, so as to automate the manual adjustments required by injecting such knowledge in the extraction algorithms themselves. Furthermore, an interesting problem to tackle is the analysis of process data that are partially on-chain and partially off-chain, to include also information sources beyond the reach of blockchains. Novel solutions could build upon the ontology-based data access approach of [8].

Acknowledgements. The authors thank Zhivka Dangarska, Dominik Haas, Jan Mendling, and Ingo Weber for their valuable support. The work of Claudio Di Ciccio was partially funded by the Austrian FFG grant 861213 (CitySPIN).

References