Validation of Smart Contracts Using Process Mining

Frank Duchmann and Agnes Koschmider

Institute AIFB
Karlsruhe Institute of Technology, Germany
frank.duchmann@student.kit.edu|agnes.koschmider@kit.edu

Abstract Smart contracts are self-executing contracts defining rules for negotiating, verifying the fulfillment of rules and executing the agreement using formal code. They run on top of a blockchain. Errors in smart contracts are costly and are mostly found too late after execution, which is too late for fixing. To improve the validation of executed smart contracts, this paper suggests a process mining based approach. For this, we present an approach for the extraction of meaningful event logs from a blockchain. The event log can be imported in any process mining tool and validation and verification techniques can be used allowing to diagnose (non)conformity in smart contracts by means of common quality measures and with low latency after smart contract execution.

1 Introduction

Blockchain is a decentralized, peer to peer network, which makes use of cryptography to securely host applications and store data, "where non-trusting members can interact with each other without a trusted intermediary in a verifiable manner" [1]. Smart contracts, which run on top of a blockchain, are self-executing contracts defining rules for negotiating, verifying the fulfillment of rules and executing the agreement using formal code. Figure 1 shows the process of a smart contract. First, a smart contract is written in a programming language (e.g.,

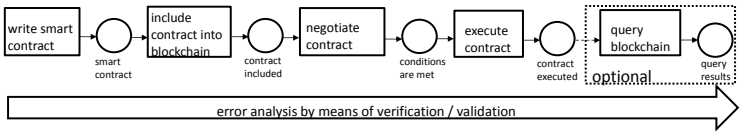

Figure 1. Process of a smart contract: from written as code to its execution, while smart contract analysis can be performed at different phases.

Solidity). Subsequently, the smart contract is deployed in the blockchain and becomes part of the blockchain. When all conditions are met, the smart contract executes itself. Verification by means of formal methods or validation by means

S. Kolb, C. Sturm (Eds.): 11th ZEUS Workshop, ZEUS 2019, Bayreuth, Germany, 14-15 February 2019, published at http://ceur-ws.org/Vol-2339

of simulation might take place in difference phases (i.e., at compile time and runtime). To analyze smart contracts after its execution, the optional activity "query blockchain" (see Figure 1) can be appended to the process referring to querying the blockchain. We motivate the need for smart contract validation by the following example. Figure 2 shows the vehicle manufacture example of a smart contract[1] within the smart contract process of Figure 1. This smart contract con-

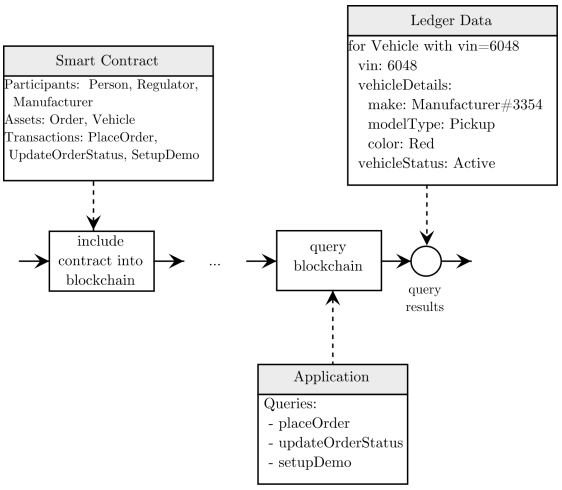

Figure 2. Example conditions for a smart contract.

sists of `Participants` with the attributes <Person, Regulator, Manufacturer>, `Assets` with the attributes <Order, Vehicle> and `Transactions` with the attributes <PlaceOrder, UpdateOrderStatus, SetupDemo>. Assume that someone aims to analyze the smart contract and query the blockchain with `placeOrder`, `updateOrderStatus` and `setupDemo` (see activity "query blockchain" in Figure 2). Further, assume that the value of `Vehicle` is changed from `SCRAPPED` to `ACTIVE` or `Order` is changed from `VEHICLEORDER_ASSIGNED` to `DELIVERED` although no prior assignment of a vehicle owner took place. The queries would only return the current value of attributes and could not detect any errors. Although such semantic errors can pass compilation and execution phases, they have to be prevented since they lead to incorrect or unexpected behavior.

2 Related Works

Related works on smart contract analysis can be distinguished by the time point of analysis. Approaches related to compile time aim to identify syntax or typechecking errors, see for reference [2]. Runtime-based analysis try to identify se-

[1] `https://www.npmjs.com/package/vehicle-manufacture-network`

mantic errors [3,4,5]. Compared to existing approaches on semantic error verification of smart contracts [3,4,5], this paper suggests a process mining approach and is superior to existing work as follows. Our approach does not require to specify additional smart contract properties as required in [3,5]. The logical flow of smart contracts is analyzed in terms of well-known Petri-net based validation and verification techniques. Also, the efficient implementation of our approaches allows to generate the process model with only low latency after the smart contract execution.

3 Process Mining based Approach to Smart Contract Validation

This next section suggests our process mining-based approach. Process mining takes event logs, records of the sequence of steps, and discovers a de-facto model of the processes. Mandatory attributes of an event log are a caseID, activity and time-stamp. Although, transactions of a blockchain are logged, the logs are of no avail for process mining due to their transactional data structure that cannot be directly mapped to an event log. We implemented the extraction of event logs from a blockchain for Hyperledger Fabric and Composer. We omitted to continue working with Ethereum due to several reasons. Currently, Ethereum does not target business smart contracts and thus the interest is low to find semantic errors. Also, too many empty blocks are mined from Ethereum, which unnecessarily expand and complicate the event log. To extract event log activities our approach relies on the queries `queryChain` and `getProcessLog`. The query `queryChain` requests from a client to register on the blockchain and returns ledger blocks as JSON objects. Note, that a direct extraction of events from the smart contract is not possible since a smart contract requires a prior execution before events can be extracted. The extracted events, thus, refer to "block writes" in the blockchain. Block writes are the part of a block that contains a list of unique IDs and the associated key value pairs that are written by a transaction[2]. The `getProcessLog` query returns block writes through iteration and transcribing each block as a trace of an event log. This means that the block writes contain the complete new record, not just the value change of attributes. Assume the attribute color of the object car1={type:pickup, color:red} is changed from red to blue, then the new block write is car1={type:pickup, color:blue}. To identify value changes, we compare the block writes. For this purpose, the last state before the change (=identical ID) is stored and compared with the current state. To ease the comparison, the possibly nested objects are transformed into a flat data structure. If a value is changed or new attributes are added, an event is transcribed. The event name consists of three attributes ranging from 1=abstract to 3=concrete, which might be exemplary eventLevel1="asset updated", eventLevel2="color changed" and eventLevel3="color changed from red to blue". Besides the eventLevel and the eventName attributes, an event-

[2] `https://hyperledger-fabric.readthedocs.io/en/release-1.3/readwrite.html`

Definition is defined, which aggregates simple events to complex events. Assume an eventLevel=“eventLevel3”, eventName=“This is now a red pickup” and eventDefinition=[color=.∗ >red] && [make=.∗> pickup]. The evaluation of the regular expression to true, is considered a complex event. The process model extracted from an exemplary smart contract is shown in Figure 3. The source code of our project can be downloaded from `https://github.com/FrankDuchmann/hf-event-extraction`.

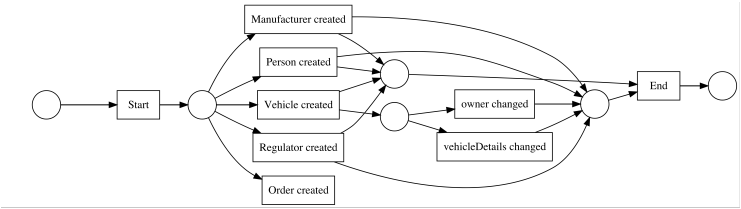

Figure 3. Process model generated from a smart contract. The process model can be validated through simulation giving hints to semantic errors in the smart contract.

4 Conclusion

This paper presented an approach for event log extraction from Hyperledger Fabric and Composer. The event logs can be processed by any process mining tool. We consider an event as the result of a transaction, which goes beyond the simplified solution of [6]. In the future we plan to target event log extraction from Ethereum.

References

1. Dannen, C.: Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners, Berkely, CA, USA (2017)
2. Amani, S., Bégel, M., Bortin, M., Staples, M.: Towards verifying ethereum smart contract bytecode in isabelle/hol. In: Proceedings of the 7th ACM SIGPLAN. CPP 2018, New York, NY, USA, ACM (2018) 66–77
3. Azzopardi, S., Ellul, J., Pace, G.J.: Monitoring smart contracts: Contractlarva and open challenges beyond. In Colombo, C., Leucker, M., eds.: Runtime Verification, Springer (2018) 113–137
4. Azzopardi, S., Pace, G.J., Schapachnik, F.: On observing contracts: Deontic contracts meet smart contracts. In: JURIX. Volume 313 of Frontiers in Artificial Intelligence and Applications., IOS Press (2018) 21–30
5. Ellul, J., Pace, G.J.: Runtime verification of ethereum smart contracts. In: EDCC, IEEE Computer Society (2018) 158–163
6. Di Ciccio, C., Cecconi, A., Mendling, J., Felix, D., Haas, D., Lilek, D., Riel, F., Rumpl, A., Uhlig, P.: Blockchain-based traceability of inter-organisational business processes. In: Business Modeling and Software Design, Springer (2018) 56–68