Towards Reusable Smart Contracts for Trustworthy Collaborative Processes (Discussion Paper)

Ada Bagozi[1], Devis Bianchini[1], Valeria De Antonellis[1], Massimiliano Garda[1], and Michele Melchiori[1]

University of Brescia, Dept. of Information Engineering
Via Branze 38, 25123 - Brescia (Italy)
a.bagozi@unibs.it, devis.bianchini@unibs.it, valeria.deantonellis@unibs.it, m.garda001@unibs.it, michele.melchiori@unibs.it

Abstract. In collaborative environments, where enterprises interact each other's without a centralised authority that ensures trust among them, the ability of providing cross-organisational services must be enabled also between mutually untrusting participants. Blockchain platforms and smart contracts have been proposed to implement trustworthy collaborative processes. However, current solutions are constrained to a specific blockchain technology and deployed on-chain the whole process, thus increasing the execution costs of smart contracts on permissionless blockchains. In this paper, we propose an approach that includes criteria to identify trust-demanding objects and activities in collaborative processes, a model to describe smart contracts in a technology-independent way and guidelines to deploy them on a blockchain. To this aim, a three-layered model is used to describe: (i) the collaborative process, represented in BPMN, where the business expert is supported to add annotations that identify trust-demanding objects and activities; (ii) Abstract Smart Contracts, based on trust-demanding objects and activities only and independent from any blockchain technology; (iii) Concrete Smart Contracts, that implement abstract ones and are deployed over a specific blockchain. Flexibility and cost reduction brought by approach are discussed in the context of a case study on remote monitoring services for the digital factory.

Keywords: blockchain, smart contract, collaborative processes

1 Introduction

In collaborative environments enterprises provide cross-organisational services to deliver integrated offerings of products and services, ensuring additional value also between mutually untrusting organisations. A real-world example is given by anomaly detection services for remote monitoring of Cyber Physical Systems in the digital factory. The Original Equipment Manufacturer (OEM) supplies the anomaly detection service, based on data streams collected from the machines of clients. In case anomalies have been identified before breakage events occur, the clients may be notified to avoid expensive repair operations, as well as costly down-times. Furthermore, insurance agencies may offer additional services to the OEM in order to limit the costs of maintenance operations under warranty, and external suppliers may know in advance scheduled maintenance interventions to prepare required spare parts. In this scenario, collected sensor data, used for monitoring and providing advanced services, must be immutable and not repudiated by clients and the implementation of anomaly detection services must be transparently shared among all participants involved in the process.

1.1 Related Work

Blockchain platforms and smart contracts have been proposed to implement collaborative processes [1, 4, 5]. In a recent research, the Caterpillar system [5] introduces an abstraction layer over the Ethereum blockchain [7] in order to support the execution of collaborative processes, represented in BPMN. Noteworthy, a process in Caterpillar is fully executed on the blockchain, coded as smart contracts, and all the states of the process instance are recorded on-chain. Moreover, the proposed solution is technology-dependent. In [3] the focus is on the benefits from the introduction of blockchain and smart contracts for the supply chain. However, the explanation about how smart contracts have been deployed is reported in a coarse-grained manner. Recent efforts also focused on the importance of conceptual modelling to demonstrate how business artefacts leverage the data-centric nature of blockchain [4] and to provide a user-centric framework to design rules deployed in smart contracts [1].

Main limitations in the current approaches are that they are technology-dependent and focus on the implementation of collaborative processes as a monolithic solution, where the whole process is deployed on-chain. This increases the execution costs on the blockchain and reduces the reuse of designed smart contracts.

1.2 Paper contribution

In [2] we proposed an approach to identify trust-demanding objects and activities in collaborative processes, a model to describe smart contracts in a technology-independent way and guidelines to deploy them in a blockchain. To this aim, a three-layered model is used to describe: (i) the collaborative process, represented using BPMN, where the business expert is supported to add annotations that identify *trust-demanding objects* and *activities*; (ii) *Abstract Smart Contracts* based on trust-demanding objects and activities only and independent from any blockchain technology; (iii) *Concrete Smart Contracts*, that implement abstract ones and are deployed over a specific blockchain, enabling the creation of a repository where a single abstract contract is associated with multiple implementations. The introduction of Abstract Smart Contracts also increases the

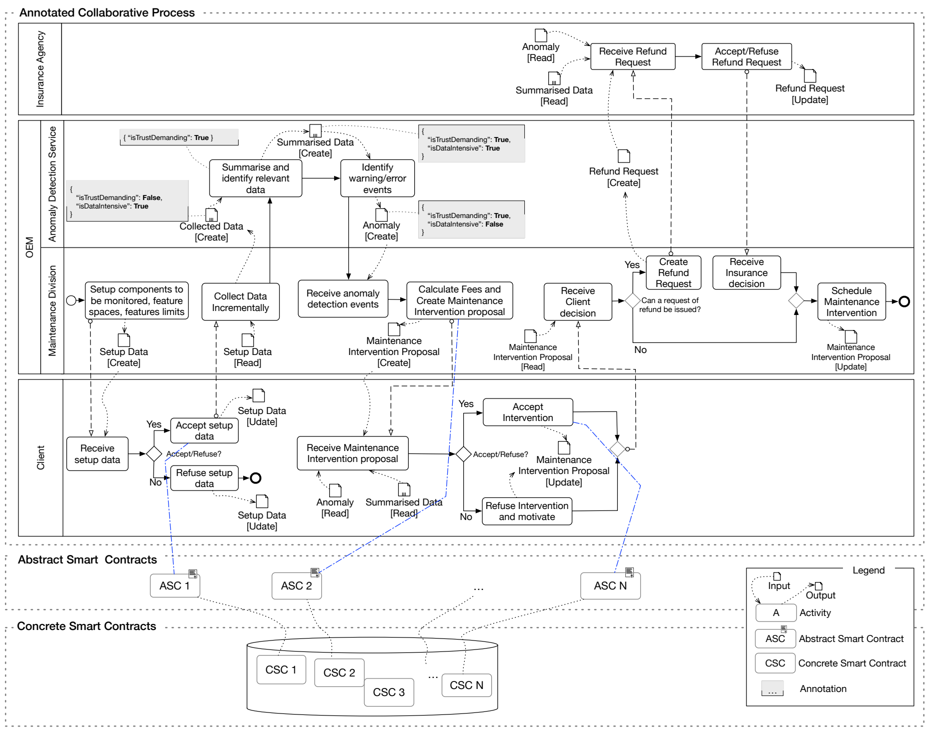
Annotated Collaborative Process
Insurance Agency
Anomaly [Read]
Receive Refund Request
Accept/Refuse Refund Request
Summarised Data [Read]
Refund Request [Update]
OEM
Anomaly Detection Service
{ "isTrustDemanding": True }
Summarised Data [Create]
{ "isTrustDemanding": True, "isDataIntensive": True }
Summarise and identify relevant data
Identify warning/error events
{ "isTrustDemanding": False, "isDataIntensive": True }
Collected Data [Create]
Anomaly [Create]
{ "isTrustDemanding": True, "isDataIntensive": False }
Refund Request [Create]
Maintenance Division
Setup components to be monitored, feature spaces, features limits
Collect Data Incrementally
Receive anomaly detection events
Calculate Fees and Create Maintenance Intervention proposal
Receive Client decision
Yes
Create Refund Request
Receive Insurance decision
Can a request of refund be issued?
No
Schedule Maintenance Intervention
Setup Data [Create]
Setup Data [Read]
Maintenance Intervention Proposal [Create]
Maintenance Intervention Proposal [Read]
Maintenance Intervention Proposal [Update]
Client
Receive setup data
Yes
Accept setup data
Accept/Refuse?
No
Refuse setup data
Setup Data [Udate]
Setup Data [Udate]
Receive Maintenance Intervention proposal
Anomaly [Read]
Summarised Data [Read]
Yes
Accept Intervention
Accept/Refuse?
Maintenance Intervention Proposal [Update]
No
Refuse Intervention and motivate
Abstract Smart Contracts
ASC 1
ASC 2
...
ASC N
Concrete Smart Contracts
CSC 1
CSC 2
CSC 3
...
CSC N
Legend
Input
Output
A Activity
ASC Abstract Smart Contract
CSC Concrete Smart Contract
... Annotation

Fig. 1. Three-layered approach overview. On the top level, is reported the BPMN of the remote monitoring services case study (only a subset of annotations is reported here).

flexibility of the approach, providing a repository of reusable software components. Furthermore, only trust-demanding activities are considered to be deployed on-chain, thus saving costs and increasing process performances.

In the following, Section 2 introduces the three-layered approach; the approach architecture will be presented in Section 3; preliminary experiments will be described in Section 4; finally, Section 5 closes the paper.

2 Three-layered approach overview

Figure 1 shows the three-layered approach proposed in this paper. Specifically, a collaborative process for Remote Monitoring Services (RMS) in the digital factory is presented. Participants of the process are the OEM, one of its clients and the insurance agency, corresponding to three different BPMN pools. The OEM pool has been further organised in two sub-partitions, Maintenance Division lane and Anomaly Detection Service lane. In our model, we do not consider trust issues between participants associated with lanes within the same pool.

Annotated collaborative process layer. At the top layer, BPMN annotations are used in order to highlight *trust-demanding objects* and *trust-demanding* *activities*. The former ones are associated to data objects (e.g., activities inputs/outputs), that are shared across different participants, and therefore transactions (i.e. Create, Read, Update, Delete actions) performed on these objects should be immutable and not repudiated by any participant. In our approach, trust-demanding objects are candidates to be permanently stored within a blockchain. Trust-demanding activities correspond to automated activities whose business logic must be non-repudiated and transparently shared among all the participants of the process. In our approach, they represent the candidates to be deployed as smart contracts within a blockchain.

Abstract Smart Contracts layer. In the middle layer, annotations are used to generated *Abstract Smart Contracts* (ASC). An ASC is described by: (i) a name; (ii) a set of state variables, that can be either primitive data types or objects, on which contract functions operate; (iii) a set of participants (i.e., users registered on the blockchain or other contracts) that are allowed to interact with the *ASC* by invoking its functions; (iv) a set of functions expressed as $functionName(IN_f) : OUT_f$, where IN_f and OUT_f are the inputs and outputs of the function. Among ASC participants, also the BPMN engine is considered, since it is in charge of invoking activities implemented as smart contracts. An ASC is generated for each BPMN activity annotated as trust-demanding. In this case: (i) the ASC name is obtained from the activity name; (ii) the set of ASC variables contains the activity inputs/outputs definitions; (iii) the set of ASC participants contains the participant of the activity and the BPMN engine; (iv) the function represents the activity, its name corresponds to the activity name, its inputs (resp., outputs) correspond to the activity inputs (resp., outputs). An ASC is generated also for each trust-demanding object as follows: (i) the ASC name is obtained from the object name; (ii) the set of ASC variables contains the object definition; (iii) the set of ASC participants contains the participant of the activity that reads the object and the BPMN engine; (iv) functions are generated according to the CRUD actions on the object. Among trust-demanding objects we distinguish those characterised by high volume and acquisition speed, denoted as *data-intensive*. Transactions on such objects, if stored within a blockchain, may require increased costs. In order to strike a trade-off between costs and tamper-proofness, the full data intensive object is kept off-chain, on top of an external Distributed File System (DFS) such as the InterPlanetary File System (IPFS), while a link to it is stored on-chain. An example of ASC is given by the trust-demanding activity `Identify Warning/Error Events` in Figure 1, where: (i) $name =$ "$IdentifyWarningErrorEventsSC$"; (ii) $variables = \{summarisedDataHash, Anomaly\}$; (iii) $participants = \{AnomalyDetection-Service, BPMNengine\}$; (iv) $function = \{identifyWarningErrorEvents-Func([summarisedDataHash]) : [Anomaly]\}$. The `summarisedDataHash` input is a hash code pointing to an external DFS on which summarised data is saved. Other ASC examples are given in [2].

https://ipfs.io

Concrete Smart Contracts Layer. At the lower layer, ASC are deployed as *Concrete Smart Contracts* (CSC) over a specific blockchain. The development of CSC starting from ASC is in charge of the developer, who has the skills to deploy CSC over a specific blockchain. The developer provides the necessary code in order to implement the CSC functions. A single ASC descriptor may be associated with multiple CSC implementations, developed for different blockchain technologies. Moreover, it separates the role of business experts, who are in charge of designing the collaborative process and being supported during the identification of trust-demanding objects and activities, and the role of developers, who possess development skills for the specific blockchain technology on which CSC will be deployed.

Example. In the multi-party collaborative process of Figure 1, proper smart contracts have been identified.

Monitoring Setup Smart Contracts (MS). Setup data is created by the Maintenance Division and read by the client, who is also in charge of accepting or declining by modifying the `Setup Data` object, that therefore is identified as trust-demanding. Therefore, an ASC is generated for the `Setup Data` trust-demanding object and all activities (associated with the Maintenance Division or the Client) that create or modify such objects are recognised as trust-demanding, bringing to the generation of corresponding ASC.

Anomaly Detection Smart Contracts (AD). Anomaly Detection Service is in charge of summarising the collection of `Collected Data` objects and applying relevance evaluation in order to identify relevant data only, generating a collection of `Summarised Data` objects. Both `Collected Data` and `Summarised Data` objects are generated at high volumes, and for this reason can be considered as data-intensive. `Summarised Data` is then used by the Anomaly Detection Service to identify warning or error events, generating an `Anomaly` object. Considering that `Summarised Data` and `Anomaly` objects are read by the Client and by the Insurance Agency, both are recognised as trust-demanding. Therefore, two ASC are generated, one for `Summarised Data` object and one for `Anomaly` object. Moreover, all activities that create or modify such objects are recognised as trust-demanding, bringing to the generation of corresponding ASC.

Maintenance Intervention Preparation Smart Contract (MIP). The `Maintenance Intervention Proposal` object is created by the Maintenance Division and modified by the Client to accept/decline it. Therefore, it is identified as trust-demanding. The same considerations hold for `Refund Request` object. All activities that create and modify these objects are denoted as trust-demanding.

3 Approach Architecture

The approach architecture is shown in Figure 2. At the bottom the architecture includes the real blockchain, on which CSC are deployed. For example, on an Ethereum network, all deployed CSC can be replicated on each client node, which hosts an Ethereum Virtual Machine (EVM). A *Log Repository* is used to store the results of every CSC execution in order to implement a communication

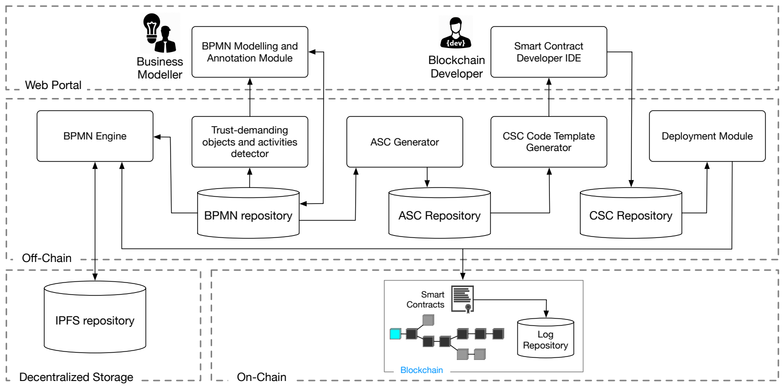

Fig. 2. Approach architecture.

channel from the smart contracts on the blockchain and external services of the collaborative process. In fact, smart contracts have no way to call external services. However, contracts can write information on a log space that is visible to external applications. A distributed storage (e.g., the above mentioned IPFS) is also considered in order to store data-intensive information without impacting on the blockchain costs.

Off-chain modules in figure implement the identification of trust-demanding objects and activities, ASC and CSC code templates generation and deployment. The BPMN is stored within the *BPMN repository*. The *ASC Generator* extracts ASC descriptors and stores them into the *ASC Repository*. The *CSC Code Template Generator* is in charge of generating templates, after choosing a specific blockchain platform. Templates are provided to the developer who completes them with code insertion. Developed CSC are stored within a *CSC Repository* and deployed on the blockchain by the *Deployment Module*. ASC/CSC repositories will contribute to increase the reuse of generated contracts. The middle layer also includes the *BPMN Engine*, that has to execute the collaborative process and must be able to invoke some activities as CSC deployed on the blockchain.

Finally, the architecture comprises a set of Web components, provided to the business modeller and the developer for editing BPMN processes, confirming or discarding suggested annotations, and editing CSC starting from the generated code templates, respectively. These, components use functionalities made available by other modules at the middle layer through RESTful APIs.

4 Preliminary Experiments

Our approach has been validated in the context of Remote Monitoring Services in the digital factory described in Section 2 and using Ethereum permissionless blockchain. However, our approach is technology-independent, and can be implemented also as a permissioned blockchain [6]. Indeed, future experiments will be focused on comparing different concrete implementations of the blockchains. Here we validated permissionless blockchain, considering that it takes more time

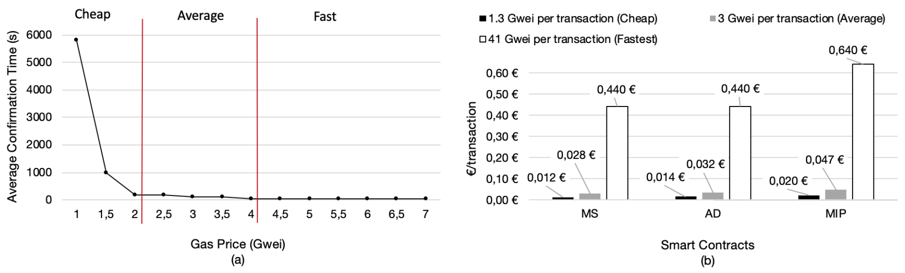

Fig. 3. Gas value required to execute all the functions of the smart contracts generated for each service in the considered case study.

to write and confirm transactions and the execution of a smart contract and the validation of a new transaction are performed by nodes that are rewarded with a fee paid by the node that initiates the transaction. Since, to the best of our knowledge, there are no collaborative processes datasets to compare our implementation with similar efforts, we performed here a cost analysis, in order to evaluate the economic effort in using a permissionless blockchain. In Ethereum-based blockchains, the cost is expressed in *gas*, which is a value established for every basic operation invoked from within the smart contract. Then, the fee paid for a transaction is expressed as the product between the gas used to process the transaction and the *gas price*, which is the amount that the node initiating the transaction is willing to pay for each gas unit.

Figure 3(a) illustrates the average time (in seconds) to confirm the transaction for each smart contract by varying the gas price, relying on the simulator provided by ETH Gas Station. As expected, the more gas price a user is willing to offer, the faster the transaction will be added to the blockchain. Figure 3(b) reports the cost per transaction required to execute each of the smart contracts implemented for the Remote Monitoring Services case study, using an Ethereum simulator. Generally, reducing the cost of the collaborative process deployment on a permissionless blockchain can be obtained by decreasing the performances of smart contract execution or limiting the elements that are deployed on the blockchain to trust-demanding objects and activity only. Indeed, our approach reduces the execution costs on the blockchain thanks to the deployment of a limited set of BPMN elements, only if necessary. In fact, only trust-demanding objects and activities will be converted in smart contracts to be deployed on the blockchain. According to the cost analysis learned by the application of our approach to the Remote Monitoring case study, as shown in Figure 3(a), from a certain amount of gas price (4.5 *Gwei*) the average confirmation time remains quite the same. Assuming that the participants have access to IPFS nodes, there is no extra cost, in terms of gas, associated with storing data-intensive objects. However, for the same application scenarios such as the anomaly detection one considered in the case study, promptness in identifying warning/error events is an

`https://ethgasstation.info/calculatorTxV.php` (accessed: July 2019)
testrpc: `https://github.com/trufflesuite/ganache-cli`

important feature of the supplied services. Therefore, this issue can be addressed by limiting the number of elements to be deployed on-chain. The proposed approach described in this paper is to meet these requirements by excluding non trust-demanding objects and activities.

5 Concluding Remarks

In this paper, we proposed a three-layered model to describe: (i) the collaborative process, represented in BPMN, where the business expert is supported to add annotations that denote trust-demanding objects and activities; (ii) Abstract Smart Contracts, based on trust-demanding objects and activities only and independent from any blockchain technology; (iii) Concrete Smart Contracts, that implement abstract ones and are deployed over a specific blockchain technology. The introduction of Abstract Smart Contracts increases the flexibility of the approach, providing a repository of reusable software components. Furthermore, only trust-demanding activities are considered to be deployed on-chain, thus saving costs and increasing process performances. Future efforts will be focused on the ASC/CSC repository. In particular, advanced matchmaking techniques (e.g. adapting Semantic Web technologies to smart contracts) will be studied to reuse existing ASC at design time or to reconciliate similar ASC. Furthemore, at runtime similar techniques will be investigated to select the proper CSC for a given ASC, depending on the cost policies implemented in different blockchains.

References

1. Astigarraga, T., Chen, X., Chen, Y., Gu, J., Hull, R., Jiao, L., Li, Y., Novotny, P.: Empowering business-level blockchain users with a rules framework for smart contracts. In: Int. Conf. on Service Oriented Computing (ICSOC 2018). pp. 111–128. Hangzhou, China (2018)
2. Bagozi, A., Bianchini, D., De Antonellis, V., Garda, M., Melchiori, M.: A three-layered approach for designing smart contracts in collaborative processes. In: 27th Int. Conf. on Cooperative Information Systems (CoopIS 2019). pp. 440–457. Rhodes, Greece (2019)
3. Casado-Vara, R., Prieto, J., De la Prieta, F., Corchado, J.M.: How blockchain improves the supply chain: case study alimentary supply chain. Procedia Computer Science 134, pp. 393–398 (2018)
4. Hull, R., Batra, V., Chee, Y., Deutsch, A., Health, F., Vianu, V.: Towards a shared ledger business collaboration language based on data-aware processes. In: Int. Conf. on Service Oriented Computing (ICSOC 2016). pp. 18–36. Banff, Canada (2016)
5. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., Ponomarev, A.: Caterpillar: A business process execution engine on the ethereum blockchain. Software: Practice and Experience 49(7), pp. 1162–1193 (2019)
6. Staderini, M., Schiavone, E., Bondavalli, A.: A requirements-driven methodology for the proper selection and configuration of blockchains. In: IEEE 37th Symposium on Reliable Distributed Systems (SRDS 2018). pp. 201–206. Salvador, Bahia, Brazil (2018)
7. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151, pp. 1–32 (2014)