Running transactional business processes with blockchain's smart contracts

Amina Brahem, Nizar Messai, Yacine Sam, Sami Bhiri, Thomas Devogele, Walid Gaaloul

Running Transactional Business Processes with Blockchain's Smart Contracts

Amina Brahem[1,2*], Nizar Messai[1†], Yacine Sam[1‡], Sami Bhiri[2§], Thomas Devogele[1¶], Walid Gaaloul[3‖]
[1]*LIFAT, University of Tours, Tours, France*
[2]*OASIS, National Engineering School of Tunis, University Tunis El Manar, Tunis, Tunisia*
[3]*Telecom SudParis, UMR 5157 Samovar, University Paris-Saclay, Paris, France*
*Email: * brahem.amina@gmail.com, †‡¶firstname.lastname@univ-tours.fr, §sami.bhiri@gmail.com,*
‖ walid.gaaloul@telecom-sudparis.eu

***Abstract*—Smart Contracts (SC for short) are gaining momentum as a suitable technology for ensuring trusted execution of Business Processes (BP for short) in open environment. Nevertheless, the transactional semantics of SC which follow ACID transactions are not appropriate for BP characteristics. Indeed, it is admitted that ACID transactions are limited to cope with complex control structure and long running execution of BP. Transactional Business Processes (TBP for short) have emerged as an extension to ACID models to overcome these limits. A TBP ensures transactional reliability of advanced transactions having a control structure as complex as for BP. In this paper, we propose an approach that builds on SC and extends them to implement TBP. We extend Caterpillar, an existing BP execution engine developed on top of Ethereum, to implement our approach and support the execution of TBP.**

***Keywords*-Business Processes; Transactional Business Processes; Open and untrusted environment; Blockchain; Smart Contracts.**

I. Introduction

The advent of the Blockchain technology with its two generations created in last years a huge amount of interest across various industries and academic communities due to its mechanisms to ensure trust in open and distributed execution environments . The first generation of blockchain allows only sending and receiving monetary values of cryptocurrencies such as the Bitcoin. While the second generation, also called the Smart Contracts (SC) blockchain, enables more complex applications like a valid car sale, a loan assessment, voting or health care tracking. A SC is simply a program enforcing the terms of the agreement between untrusted parties.

SC are gaining momentum as a suitable technology for ensuring trusted execution of Business Processes (BP for short) and BP collaborations in open environment [1], [2]. Nevertheless, the transactional semantics of SC are the same as in ACID models [3], [4]. Consequently, we argue that bare SC are limited to support transactional requirements of BP executions. Indeed, it is admitted that ACID transactions are limited to cope with complex control structure and long running execution of BP [5], [6]. Besides, the rollback mechanism in BP is not as obvious as for conventional database operations.

Transactional Business Processes (TBP for short) [6], [7] have emerged as an extension to ACID transactions to overcome these limits. More precisely, TBP bring together Business Processes and Advanced Transactional Models [5], [8]. For the former, they ensure transactional correctness. For the latter, they support large flow-based applications having complex control structure. A TBP defines a transactional flow on top of the control flow. The transactional flow englobes transactional properties of the BP activities (pivot, compensatable, retriable) and transactional mechanisms. The latter include recovery mechanisms such as compensation as backward recovery and alternatives as forward recovery. This definition of the transactional flow was largely inspired from one of the Advanced Transaction Models which is the Flexible Transactional Model [5].

In this paper, we propose an approach that builds on SC and extends them to implement TBP. We extend Caterpillar [9], an existing BP execution engine developed on top of Ethereum, to implement our approach and support the execution of TBP. Caterpillar takes a Business Process Model Notation (BPMN) 2.0 [1] specification as input and generates the corresponding control flow in a SC.

Our approach implements a TBP where its control flow is given as input and its transactional flow follows the Flexible Transactional Model principles. The generated SC implements both the control flow and the transactional flow.

For the implementation of our approach, we use scenarios related to touristic trip planning.

The remainder of this paper is structured as follows. A motivating scenario is presented in Section 2. In Section 3, we introduce background concepts. We detail our approach in Section 4. Section 5 discusses the related work. Finally, we conclude in Section 6.

II. Motivating scenario

We consider a motivating scenario of a tourist who wants to order a personalized package for her/his visit to the Château of Tours [2]. However, the choices offered to the tourist in terms of the "ways" to visit the châteaux are

[1]BPMN 2.0 standard definition: https://www.omg.org/spec/BPMN/2.0
[2]https://www.loirevalley-france.co.uk/loire-chateaux

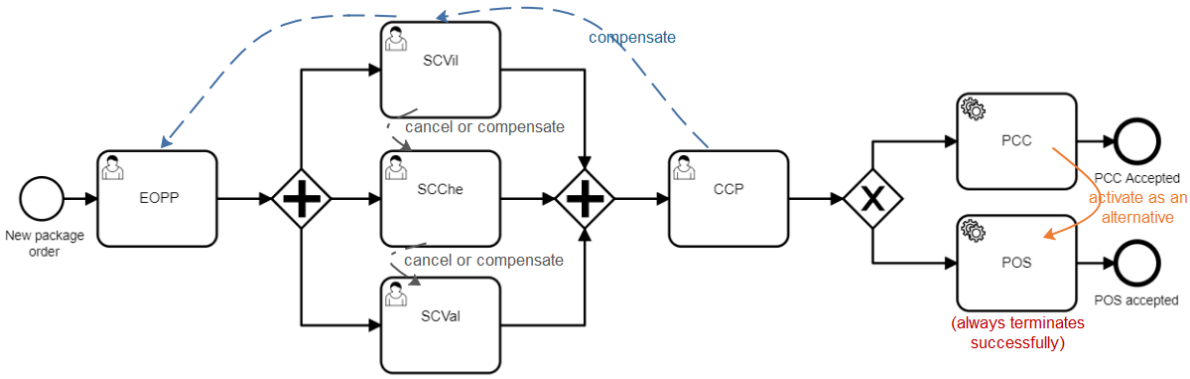


Figure 1. Personalized touristic package ordering process. *In colors: the transactional flow*

limited. She/he either buys individual tickets or chooses one of the very few predefined packages at the available web sites and which may not entirely reach her preferences. As a matter of fact, offering a customized or personalized service is a concept which is more and more used in the domain of tourism, especially in trip planning [10]. Here, we model a scenario of a tourist who wants to visit the châteaux "along the river of the Cher" and thus designs and buys her personlized package. She enters order for a personnalized package (EOPP). She selects the châteaux that she wants to be included in the package such as the "château of Villandry" (SCVil), the "château of Chenonceau" (SCChe) and the "château of Valençay" (SCVal). Then, after confirming her choice of the package (CCP), she pays using one of the available payment methods: credit card (PCC) or paying on-the-spot (POS).

If we try to add a valid transactional flow to the presented model according to transactional semantics in the Flexible Transactional Model (see Figure. 1), when PCC fails, POS will be activated as an alternative and the process ends in a consistent state given the fact that POS always terminates successfully. The transactional flow contains as well the following failure recovery mechanisms: (i) in case of failure of CCP, the work already done has to be compensated, i.e., selection of the package components as well as EOPP;
(ii) if SCVil fails, it will cancel the other tasks running in parallel or compensate them if they have already completed their work.

To motivate our choices of the used technologies, first, the present scenario involves multiple participants, i.e., the tourist and payment services. Thus, we are in the case of a distributed execution where there is a need for trust and here comes our choice of SC technology that alleviates the need for a central orchestrator and ensures trust. For (partial) failure handling and ensuring transactional correctness, we use the transactional mechanisms embedded in the transactional business processes, particularly in the flexible transactional model. In the next section, we give an overview of principal concepts related to our work.

III. Background Concepts

A. Blockchain and Smart contracts

A blockchain [11] is a distributed ledger where data is organized in sequence of blocks equivalent to containers of transactions. The blockchain network is maintained by independent computers referred to as nodes or peers that do not know or trust each other but can connect and cooperate to validate transactions executed on the blockchain. After validation, peers record, share and synchronize transactions in their respective electronic ledgers. The first generation of blockchain allows only sending and receiving monetary values (e.g., Bitcoin).

In the SC blockchain, meanwhile, a transaction allows more complex operations such as the creation of a SC and stores the results of function calls in SC. A SC can be defined as a computer program enforcing the terms of the agreement between untrusted parties about a valid car sale or a loan assessment or voting or health care tracking, etc. Any user can create a SC by posting a transaction in the Blockchain. Once created, the contract is assigned to a unique address (a secure identifier) used to interact with that particular contract [12]. A SC is written in a SC-specific programming languages like Solidity for Ethereum. It is then compiled into 'bytecode' that is read and run on the 'ethereum virtual machine' (EVM). A SC can define multiple entry points of execution (function calls). After a contract finishes processing a message it receives, it can send back a return value to the sender. The contract's state will then be updated accordingly.

SC cannot exchange data with off-chain world directly. Henceforth, Oracles are of common use in this case. Oracles are real-time data feeds that act as mediator between SC and external world. In more details, oracles forward a request from external application to the SC via a Solidity event and receive corresponding response via a contract function call.

We can trust the entire system of the blockchain because, first, everyone in the blockchain keeps a (partial or full) copy of the chain and can check what is exactly happening in the blockchain. Furthermore, transactions on the blockchain are agreed i.e., a transaction is accepted when the majority of the network have a concensus on its validity. Finally, transactions are immutable, they cannot be revoked and state changes cannot be undone. Trust in the correct execution of SC extends directly from regular transactions since their creation function calls result from blockchain transactions.

B. Transactional Processes

The transactional approach initially emerged in the context of database management systems with ACID models. The attractiveness of this approach is due to its simplicity and systematic way to handle failures. This lead to its adoption to the context of flow-based applications. However, ACID models turned to be restrictive to support such applications [5], [6]. The latter may have more complex control structure than a simple sequence of read/write operations and a long running execution. Besides, the rollback mechanism is not as obvious as for conventional database operations.

Advanced Transaction Models (ATM)

ATM extend the definition of flat ACID transactions. They relax the atomicity property to define instead the "failure atomicity" or partial failure of a process activity [8]. In addition, they relax isolation property to allow high inter-process concurrency handling and compensation as backward recovery.

Later appeared the Flexible Transactional Model (FTM) [5] which implements many transactional mechanisms at one time. The FTM is appropriate to business processes with complex control structure. It includes many transactional mechanisms: (i) it allows failure atomicity; (ii) it permits alternatives as forward recovery ; (iii) compensation as backward recovery and (iv) assigns termination properties to process activities (compensatable, pivot, retriable). Let's come back to our motivating scenario illustrated in Figure. 1:

- SCVil is **compensatable** which means that its undesirable effects can be reversed after completion through the execution of its compensation activity. The latter is an additional activity that serves to reverse the effects of the main compensatable activity.
- CCP is **pivot**, i.e., it (a) completes successfully or (b) fails and compensate the work already done. Once completed successfully, it is a no-return point in the BP execution.
- POS is **retriable**, i.e., it will surely terminate with success after a certain number of $\{failure \rightarrow reactivation\}$ attempts.
- An activity can be also **effect-free** when it is considered that its completion has no effect to be compensated.

Transactional Business Processes (TBP)

Transactional Business Processes (TBP for short) bring together business processes and Advanced Transactional Models [6], [7]. For the former, they ensure transactional correctness. For the latter, they support large flow-based applications with having complex control structure. A TBP has two perspectives: the control flow and the transactional flow. The control flow describes the order in which process activities are activated as well as an execution preference order among alternatives.

The transactional flow englobes a set of transactional properties assigned to business process activities. Transactional properties are the equivalent of termination properties in the FTM, i.e., **compensatable**, **pivot** and **retriable**. The transactional flow describes as well the different transactional mechanisms to enable a BP either to complete successfully or fail "properly". Following an activity failure, the transactional flow: (i) tries first to execute an alternative if it exists; (ii) otherwise in case of a fatal failure (i.e., no alternative) causing the overall BP failure it compensates the work already done and (iii) cancel all running executions in parallel.

In the next Section, we present the proposed approach consisting in extending bare SC to support the execution of Transactional Business Processes.

IV. Smart Contract-based Transactional Business Process: Design principles

Our approach builds on an existing BPM system developed on top of Ethereum plateform called Caterpillar [9]. Caterpillar takes a BPMN 2.0 specification as input and generates the corresponding control flow in a SC.

We extend Caterpillar in order to implement in addition the transactional flow. Our approach implements a Transactional Business Process (TBP) where its control flow is given as input and its transactional flow follows the Flexible Transactional Model principles. The generated SC implements both the control flow (Section IV-A) and the transactional flow (Section IV-B).

A. Implementing the control flow

Caterpillar logic for implementing the control flow relies on simulating the token game and defining treatment to execute for each state of an activity's life cycle [1]. The life cycle of a process activity is: *enabled* $\rightarrow$ *started* $\rightarrow$ *completed*.

To do so, Caterpillar uses a Solidity code generator called "BPMN-to-Solidity template" that generates the Solidity code of the called **"Process SC"** from a given BPMN specification. Depending on the position of the token in the process, the template generates predefined functions to execute in the "Process SC".

Caterpillar uses as well a second Solidity code generator which is the "Oracle-to-Solidity template". This template produces the code of the **"Oracle SC"**. The Oracle SC is associated to the Process SC and handles the interaction of the latter with external users and applications. For instance, it publishes a Solidity event in the Ethereum log when a user task is started. It also can read data from the Process SC to return to the external user.

B. Implementing the transactional flow

We build on the logic of Caterpillar tool to implement the transactional flow. First, we assign transactional properties to process activities and enrich process activities life cycles with transactional states in order to model their transactional properties. Then, we modify the Solidity code generation templates (BPMN-to-Solidity and Oracle-to-Solidity templates) in a way that, depending on the transactional properties of the activity, the templates generate functions to execute for each of the transactional states.

1) Extending activity life cycle: Each process activity has one or many transactional properties, i.e., pivot, compensatable and retriable. The only termination state of an activity in Caterpillar is *completed*. We add in addition four other possible termination states which are *aborted*, cancelled, *failed* and compensated which we call transactional termination states.

A **pivot** activity (Figure. 2a) (similarly to **compensatable** and **retriable** activities) is initially at the state *enabled*. Then, some event may occur and lead to an early abortion (state *aborted*) or nothing happens and the activity is *started*. During its execution, an activity can be *cancelled*. There are next two possibilities: (a) the activity completes successfully and reaches the state *completed*; (b) it fails and meets the state *failed*. In addition to that, a **compensatable** activity effect could be *compensated* (Figure. 2b). Whereas a **retriable** activity can be reactivated after failure via the *retry()* transition (Figure. 2c). Finally, an activity can combine two transactional properties like being **compensatable and retriable** (see Figure. 2d).

2) Implementing transactional mechanisms: We bring the following modifications to the Solidity code generation templates. In this way, the "Process S.C" and the "Oracle S.C" are enriched to implement the transactional mechanisms upon the extended life cycle:

(i) We define for each process task called **ActivityName**, a state vector named **StateVector_ActivityName** to capture the activity state's evolution in the "Process S.C". StateVector_ActivityName is a bit array where each bit encodes a different state that an activity can reach. A bit array is encoded as a 256-bits unsigned integer, which is the default word size in the Ethereum Virtual Machine.

(ii) We generate functions corresponding to the four transactional termination states. For instance, a function called **ActivityName_fail()** is generated. This function handles the activity's subsequent behavior in case it fails. Similar functions are defined in the case of an activity's abortion, cancellation and compensation.

(iii) We implement the transitions (*abort()*, *cancel()*, *complete()*, *fail()* and *retry()*) which allow the activity to move from one state to another as functions in the "Process S.C". These functions update the value of activities state vectors.

(iv) We add also endorsement for events in the "Oracle SC" to abort, compensate or cancel an activity.

Due to lack of space, we do not provide a description of the implementation of the approach. We host the source code at https://github.com/TBP-BE/TransactionalBP-BlockchainExecution. The source code repository contains as well the code generated for the motivating scenario in Section II of ordering a personalized package.

V. Related Work

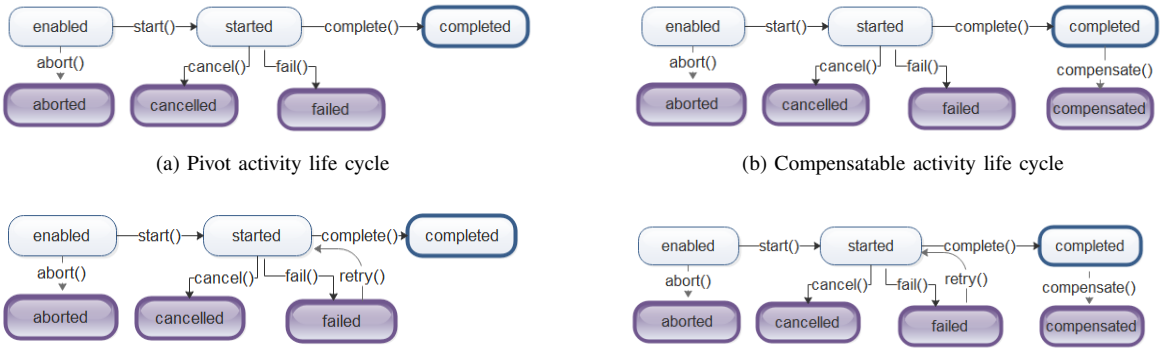
Recent works use the blockchain as a coordination mechanism to implement the control flow of collaborative processes. The current process state is stored on the chain and the execution of the next process task depends on the process data shared among the network peers.

In [13], a runtime verification mechanism is developed to verify running choreographies of process instances on top of Bitcoin blockchain. The approach is applicable only to standard Bitcoin network transactions like Pay-to-Script-Hash, over and above, it does not address fault management and recovery. Authors make some propositions like the participant that starts the process execution can intercept the process token and collect it if something goes wrong with the process execution.

In [1], authors present an approach to enable the monitoring and execution of collaborative processes using blockchain's SC. The approach is based on a translator taking as input the business process model in Business Process Model Notation (BPMN) and generates the corresponding contract according to specific BPMN-to-Solidity code transformation rules. A tool named Caterpillar [9] is derived from the approach in [1]. It is an open-source Blockchain-based business process execution engine that translates the business processes modeled in BPMN into SC written in Solidity programming language. Caterpillar supports also the execution of collaborative processes.

In a more recent work [14], authors start from a declarative input (XML) and propose a set of transformation rules to convert touristic itineraries presented in XML to SC.

However, the proposed approaches [1], [14] as well as the existing tools like Caterpillar [9] handle only the workflow execution of business processes without taking into consideration the case of task failures and the subsequent behaviour of the business process in such cases.



(a) Pivot activity life cycle

(b) Compensatable activity life cycle

(c) Retriable activity life cycle

(d) Compensatable and retriable activity life cycle

Figure 2. Activity life cycle according to the different transactional properties. [6]

VI. Conclusion

Motivated by the opportunities offered by the technology of SC to enable the trusted and decentralized execution of BP in open environments and the TBP to handle failures and ensure the transactional correctness of BP executions, we propose an approach that builds on SC and extends them to implement TBP. Our approach implements a TBP where its control flow is given as input and its transactional flow follows the FTM principles. The transactional flow englobes the transactional properties assigned to process activities as well as a set of transactional mechanisms (compensation and enabling alternatives).

In order to implement our approach, we extend an existing blockchain-based BPM system named Caterpillar in order to support the execution of TBP. Caterpillar generates only the control flow of input BPMN models. We mainly modified the Solidity code generation "templates" of the tool in order to implement in addition the transactional flow.

We consider the present work as a first step to providing a blockchain-based execution environment for TBP. An avenue for future work is to examine the cost of execution of our approach especially in case of large BPs. Finally, we intend to consider permissioned blockchains since they ensure better performance and privacy and (arguably) lower operational costs.

References

[1] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, "Untrusted business process monitoring and execution using blockchain," in *International Conference on Business Process Management*. Springer, 2016, pp. 329–347.

[2] J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar *et al.*, "Blockchains for business process management-challenges and opportunities," *ACM Transactions on Management Information Systems (TMIS)*, vol. 9, no. 1, p. 4, 2018.

[3] R. Hull, V. S. Batra, Y.-M. Chen, A. Deutsch, F. F. T. Heath III, and V. Vianu, "Towards a shared ledger business collaboration language based on data-aware processes," in *International Conference on Service-Oriented Computing*. Springer, 2016, pp. 18–36.

[4] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.

[5] H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek, "Atomicity and isolation for transactional processes," *ACM Transactions on Database Systems (TODS)*, vol. 27, no. 1, pp. 63–116, 2002.

[6] S. Bhiri, W. Gaaloul, C. Godart, O. Perrin, M. Zaremba, and W. Derguech, "Ensuring customised transactional reliability of composite services," *Journal of Database Management (JDM)*, vol. 22, no. 2, pp. 64–92, 2011.

[7] J. El Hadad, M. Manouvrier, and M. Rukoz, "Tqos: Transactional and qos-aware selection algorithm for automatic web service composition," *IEEE Transactions on Services Computing*, vol. 3, no. 1, pp. 73–85, 2010.

[8] M. Rusinkiewicz and A. Sheth, "Specification and execution of transactional workflows," in *IN MODERN DATABASE SYSTEMS: THE OBJECT MODEL, INTEROPERABILITY, AND*. Citeseer, 1994.

[9] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, and A. Ponomarev, "Caterpillar: A business process execution engine on the ethereum blockchain," *Software: Practice and Experience*, vol. 49, no. 7, pp. 1162–1193, 2019.

[10] M. Boulakbech, N. Messai, Y. Sam, and T. Devogele, "Visual configuration for restful mobile web mashups," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 870–873.

[11] S. Underwood, "Blockchain beyond bitcoin," *Communications of the ACM*, vol. 59, no. 11, pp. 15–17, 2016.

[12] C. Dannen, *Introducing Ethereum and Solidity*. Springer, 2017.

[13] C. Prybila, S. Schulte, C. Hochreiner, and I. Weber, "Runtime verification for business processes utilizing the bitcoin blockchain," *Future Generation Computer Systems*, 2017.

[14] A. Brahem, N. Messai, Y. Sam, S. Bhiri, T. Devogele, and W. Gaaloul, "Blockchain's fame reaches the execution of personalized touristic itineraries," in *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, 2019, pp. 186–191.