

Temporal Constraints in Smart Contract-Based Process Execution: A Case Study of Organ Transfer by Healthcare Delivery Drone

Amal Abid¹, Saoussen Cheikhrouhou^{1,2}, Slim Kallel^{1,2} and Mohamed Jmaiel^{1,2}

¹ReDCAD, National Engineering School of Sfax, University of Sfax, Sfax, Tunisia

²Digital Research Center of Sfax, Sfax, Tunisia

Abstract

Blockchain has emerged as one of the most promising technologies in Industry 4.0. It has the potential to improve interorganizational processes, especially by addressing lack-of-trust issues. Recently, sophisticated Blockchain-based BPM systems have been proposed to enable the execution of business processes in the Blockchain. Despite the importance of the aspect of time in process modeling, these approaches do not provide methods to represent nor to manage temporal constraints for business processes. This insufficiency is due to inherent limitations of Blockchain platforms which do not offer native means of measuring time. We have proposed an approach that enables the transformation of a large set of temporal constraints from a business process model to a smart contract code. In this paper, we aim to validate the feasibility and efficiency of our prior work through a case study of organ transfer by healthcare drone delivery.

Keywords

Blockchain, BPM, Healthcare Drone Delivery, Organ Transfer, Temporal Constraints

1. Introduction

In today's business world, organizations are likely to experience new challenges on account of the evolution of modern technologies such as Cloud Computing, Cyber-Physical Systems, Internet of Things, and Blockchain in Industry 4.0. Particularly, Blockchain has attracted high interest as a distributed storage and computing technology for building the next generation of applications. Indeed, Blockchain-based smart contracts have the potential to perform decentralized, transparent, and inviolable process execution which is relevant to address lack-of-trust issues in inter-organizational processes [1].

Some advanced work has emerged along this direction, enabling modelling and execution of business processes (BPs) in the Blockchain [2, 3, 4]. However, these approaches usually omit the essential aspect of time, which is of paramount importance in both process models [5] and contractual agreements. More precisely, temporal constraints such as deadlines have


Tunisian Algerian Conference on Applied Computing (TACC 2021), December 18–20, 2021, Tabarka, Tunisia

✉ amal.abid@redcad.org (A. Abid); saoussen.cheikhrouhou@redcad.org (S. Cheikhrouhou); slim.kallel@redcad.org (S. Kallel); mohamed.jmaiel@redcad.org (M. Jmaiel)

🆔 0000-0003-0669-8406 (A. Abid); 0000-0003-4607-7452 (S. Cheikhrouhou); 0000-0002-2824-167X (S. Kallel); 0000-0002-2664-0204 (M. Jmaiel)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

to be properly considered by different stakeholders to accomplish a common business goal. Although process modeling languages usually offer a variety of elements to express such temporal constraints [5], Blockchain platforms do not provide methods to represent nor to manage temporal constraints for BPs. More specifically, transaction completion time is not fixed, it can take from a few seconds to several minutes. This insufficiency is due to mining protocols, inherent limitations of smart contracts, and the absence of a native means of measuring time on common Blockchain platforms. Consequently, when executing some BP's tasks/ activities via a smart contract code, it does not offer any means to check whether BPs' temporal constraints are respected or not.

In our prior work [6], we have extended Caterpillar [2], which is an open-source blockchain-based BPMN execution engine, to enable the transformation of a large set of temporal constraints from a business process model to a smart contract code. This prior work is the first research attempting to consider BPs temporal constraints in Blockchain smart contract code. However, it lacks the validation of its feasibility and effectiveness. This paper aims to validate the prior work through a suitable case study. We consider organ transfer by healthcare drone delivery, which is a very time-sensitive case study. All involved actors and assets such as emergency blood supplies, vaccines, medicines, diagnostic samples, and organs, are constrained by hard timing requirements. These requirements are of paramount importance and should be considered from the beginning of the process modeling step. For example, the flight time is a strict requirement for drones where a missed deadline may lead to critical situations. Statistics given in [7] show that 20% of emergency patients' deaths were caused by traffic jams in traditional road transport infrastructure. Besides, drones can be preferred for medical supplies, since an ambulance drone can perform a 93% faster response time in rural areas and 32% in urban areas than conventional methods [8]. For organ transfer, in particular, an organ should be harvested from the donor as soon as possible, since it becomes progressively less healthy over time. Therefore, the transfer time is decisive to the success of the surgery. Moreover, about 1.5% of donor organ transfers did not reach their target destination, and about 4% had an unexpected delay of more than two hours, according to the United Network for Organ Sharing [9]. The proposed approach takes properly into account a wide range of temporal constraints, reducing thus organ transfer time to just a few minutes instead of several hours or even days.

To prove the feasibility and efficiency of the proposed approach, this paper proceeds the following steps : (i) Modelling and configuring the business process, (ii) Generating and deploying smart contracts, (iii) Creating and executing a process instance, and (iv) Evaluating the proposed solution with respect to financial cost.

The rest of this article is organized as follows : Section 2 briefly introduces some concepts upon which our work is built. Section 3 provides an overview of the organ transfer case study. Section 4 illustrates the modelling and configuration of the organ transfer process. Section 5 details process's generation and execution. Section 6 evaluates our approach. Section 7 summarizes related work. Section 8 concludes and suggests future directions.

2. Background

This section introduces the main concepts and definitions related to Caterpillar and the Temporal Constraints extension.

Caterpillar is a blockchain-based process execution engine that runs on top of Ethereum. Caterpillar is mainly composed of a Compilation Tools module, an Execution Engine, and an Execution Panel as shown in Figure 1. Caterpillar's Execution Engine allows the deployment of smart contracts as well as the creation and execution of process instances in the Blockchain, while Caterpillar's Execution Panel enables the tracking of process instances graphically.

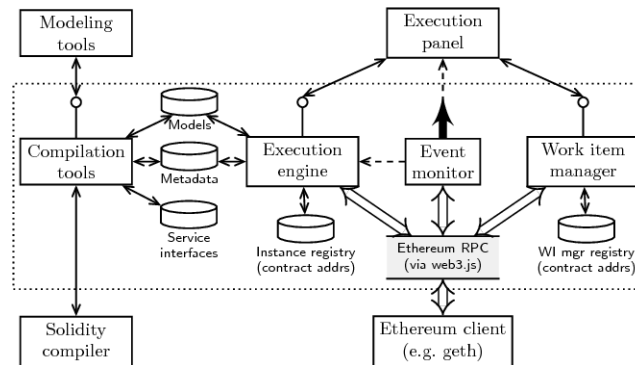


Figure 1: Architecture of Caterpillar [2]

The “Compilation Tools” module of Caterpillar establishes a comprehensive BPMN-to-Solidity mapping. Given a BPMN model in standard XML format, this module generates a smart contract in Solidity, which encapsulates the workflow routing logic of the process model. Caterpillar supports not only basic BPMN control flow elements (i.e. tasks and gateways), but also includes advanced ones, such as subprocesses, multi-instances and event handling. However, we denote that Caterpillar does not support temporal constraints on BPs. Consequently, the generated smart contract code could be undetermined, and thus temporal constraints could be violated. Our prior work [6] extends the Caterpillar compiler by several temporal constraints like duration, temporal dependency, temporal constraint over cardinality, and start/end temporal constraints. In order to translate BPMN model to smart contract code we must proceed with the following mapping rules of Caterpillar. (i) Marking variable: A variable that encodes the overall distribution of tokens across the sequence flows, (ii) Step function: A function that allows making steps during a whole business process execution, and (iii) Bit-wise operations: Operations that are used to ensure all the queries/updates on the process state.

To extend Caterpillar, we have proposed our mapping rule for the transformation of a BPMN model containing Temporal Constraints to solidity code. This transformation contains a Time variable, a Time function and a time guard to keep harmony with the mapping rules of the Caterpillar compiler, which are detailed as follows. (i) Timer variable: When a process model holds a timer event or an advanced temporal constraint, a corresponding variable will be generated at the creation of the contract, (ii) Timer function: This function will be invoked by the activity to check its conformance with its temporal constraint, (iii) Timer guard: An

extra timer guard will be generated to basically check timestamps and thus meet the timer requirements. Figure 2 illustrates the proposed extension through an example of the duration constraints. We refer the reader to [2, 6] for more details.

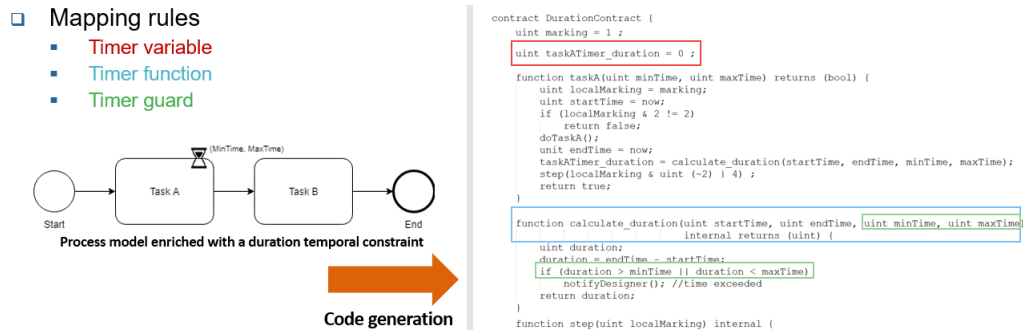


Figure 2: Temporal Constraints extension for Caterpillar

3. Organ transfer case study overview

This section presents an overview of the organ transfer case study, which implies various safety and effectiveness considerations including medical and air traffic issues. By harnessing a Blockchain-based infrastructure for healthcare drone delivery, the proposed approach ensures trusted information availability, which is crucial in both aviation processes and medical standards. Besides, the proposed approach takes into consideration additional complexity of rapidly evolving, sensitive, and time-critical processes as would be the case for an organ transfer by drone. These processes include drone data recording on the Blockchain, efficient drones' communication, and drone crash analytics (see Figure 3).

In addition, organ transfer processes imply different actors as well as various types of interdependencies. Involved actors include the Air Traffic Control (ATC), the drone operator, the drone enclosing the organ container, the organ-supplying hospital, and the receiving hospital. Each actor requires different types of access to the Blockchain (i.e., read from the Blockchain, write to the Blockchain, or both). The data maintained securely on the Blockchain includes the drone's position (e.g., course, speed, altitude, etc.), the drone's intent (i.e., planned/ subsequent route and altitudes), the drone's status (e.g., fuel level or state of charge, etc.), the organ itself (e.g., the donor's blood type and age, time of harvesting, etc.), and the organ container on board the drone (e.g., the container's temperature, humidity, vibration level, etc.). Table 1 clearly illustrates the relation between involved actors, access types and data stream.

Organ transfer processes are detailed as follows :

- **Data recording on the Blockchain:** An Organ delivery drone can store key sensor and flight telemetry on the Blockchain. This data is encrypted and only available to the appropriate actors. Using the Blockchain as a storage of all drone telemetry data, we can maintain a chronological record of events on-chain. Hence, all authorized actors in the delivery process can access this data. Furthermore, an organ delivery drone can only perform actions if appropriate

Table 1

Data and access types required by all actors

Data maintained on the Blockchain	Actors need to read some data from the blockchain	Actors need to write some data on the blockchain
Position	ATC, Operator, Receiving hospital	Drone
Intent	Drone, ATC	Operator
Drone status	Operator	Drone
Organ	Receiving hospital	Organ-supplying hospital
Container	Receiving hospital, Organ-supplying hospital, Operator	Drone

permission is given in the form of a signature on-chain by the operator that owns it, authorizing only that operator to control the drone.

- **Secure and efficient Drones' Communication through the Blockchain:** For the purpose of coordination, the communication with other drones can occur publicly. Here, the Blockchain can serve as a distributed communication for drones. Other aircrafts can also interface with the Blockchain to view the blocked/ open zones and if any issues have occurred during flights that other aircraft need to be aware of. Therefore, drones, regardless of their operators can access a shared pool of flight and airway information on the Blockchain. This pool of information is specific to the area in which the drone is currently operating. This allows the drone to be aware of updated information such as telemetry and environmental conditions, and thus can make traffic adjustments, decrease flight time and enhance the management of temporal constraints. This would be interesting specifically in case of organ transfer.

- **Drone Crash Analytics:** As the number of drones flying in the skies has increased, accidents will be inevitable. Insurance companies will need access to historical drones' flight data to deduce accident reasons. A drone can record all its flight details on the Blockchain, which keeps the traceability and helps thus to track any crash or malfunction, similarly to the blackbox on an airplane. This data can then be harnessed to enhance functionality as well as safety in future drone deliveries.

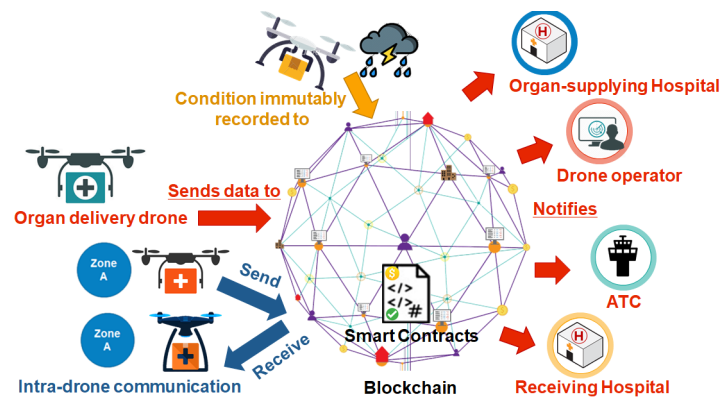


Figure 3: Organ transfer case study overview

4. Modelling and configuration of the organ transfer process

Figure 4 shows a simplified BPMN model of the organ transfer case study. Five actors are represented: (1) Organ-supplying hospital, (2) Drone, (3) Drone operator, (4) ATC, and (5) Receiving hospital. Initially, the organ-supplying hospital provides some data about the organ to transplant in order to mitigate the likelihood of medical errors at the receiving hospital. This data includes the donor's blood type and age as well as the time of harvesting. Besides, the organ-supplying hospital prepares organ container requirements and requests for the drone flight. Once the drone takes flight, it regularly sends both flight telemetry and environmental conditions (e.g. each 5 minutes). Consequently, the drone operator decides to update the flight path of the drone based on newly updated environmental conditions data. The drone notes the new flight plan and adjusts flight path and altitude. Once the Air Traffic Control is aware of the new flight path, it makes traffic adjustments accordingly. Meanwhile, the drone's organ container adapts its internal pressure, humidity, temperature and vibration level data when reaching new altitudes. The drone periodically updates organ container data, allowing the receiving hospital to monitor the health of the organ based on updated data. Afterwards, the drone updates the time of arrival. The organ must be delivered as soon as the organ is harvested. Otherwise, the organ health would degrade dramatically. Therefore, the drone must deliver the organ before the deadline. If the deadline is exceeded, the drone must report the reasons of delay to enhance future drone deliveries. Finally, once the organ is delivered, the receiving hospital can adjust the surgery schedule according to new arrival time due to flight plan change.

Within this case study, the temporal perspective is critical since temporal constraints must be respected. The associated temporal constraints are as follows :

- The flight time should not exceed a given deadline (e.g. 30 minutes). The drone should deliver the organ as soon as possible while respecting a given duration (e.g. between 5 and 30 minutes), since the organ health would degrade dramatically over time.
- The duration of each synchronization task (e.g. "Send flight telemetry and environmental conditions", "Update organ container information", "Update the flight path", "Make traffic adjustments", "Update the arrival time") is between 15 and 30 seconds. This duration is affected by Blockchain congestion.
- Some activities have a recurrent behaviour that must be repeated every 5 minutes and no more than five times. These activities include "Send flight telemetry and environmental conditions", "Update organ container information", "Update the flight path", "Make traffic adjustments", and "Update the arrival time".

The "Modeling Tool" of Caterpillar is developed on top of Camunda's BPMN modeler. To make the models executable, some configurations should be applied. In particular, code needs to be added such as process variable definitions, Solidity codes for script tasks, and codes identifying the information exchange (i.e. from/to user/service tasks). For example, to indicate that an organ-supplying hospital must provide donor's health information via the task, a snippet should be added in the BPMN documentation element under this task (see Listing 1). This snippet indicates that an organ-supplying hospital must provide donor's health information (e.g. via a web form as shown in Figure 6b in section 5.2). The second part of the snippet is used

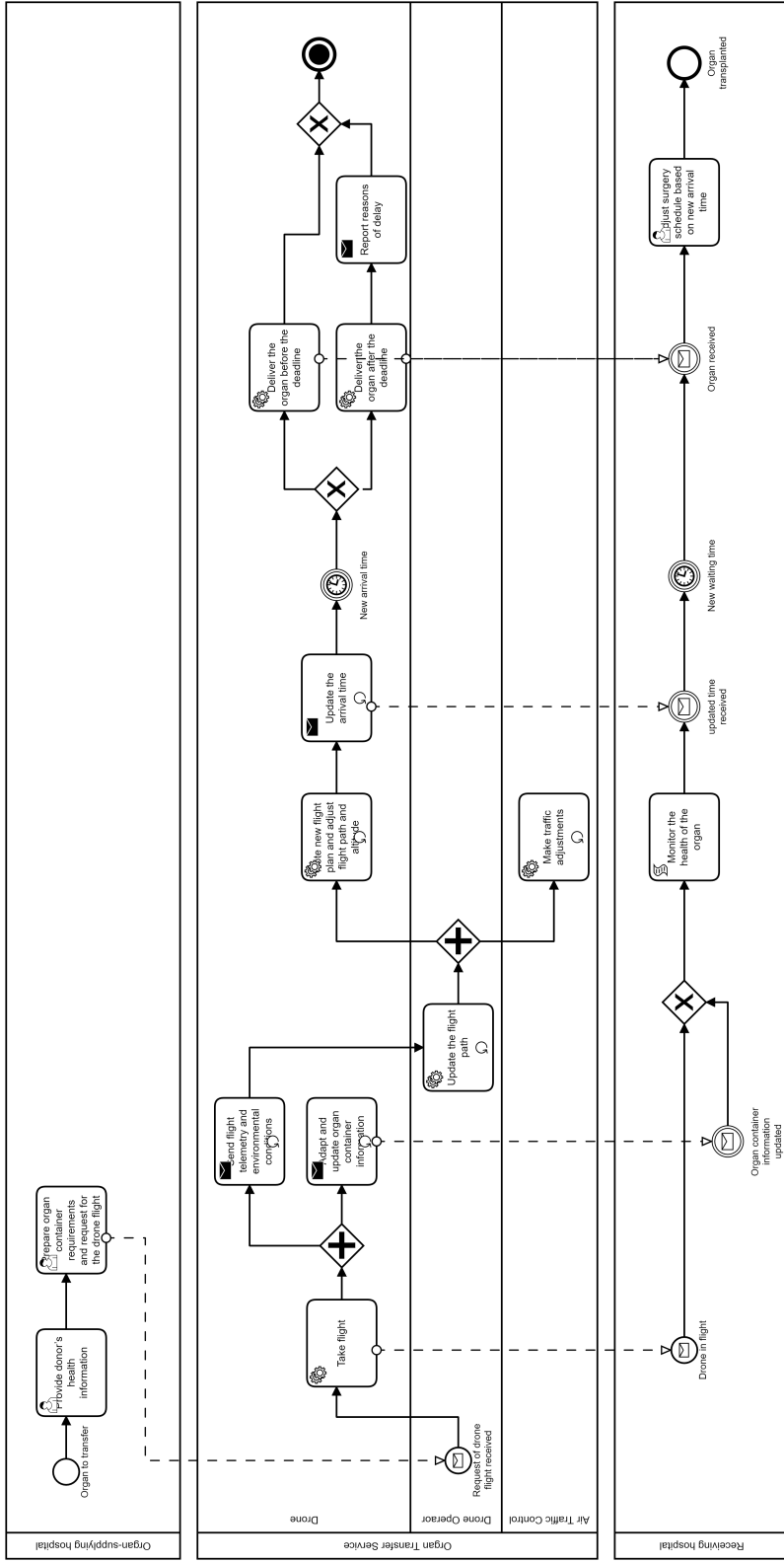


Figure 4: Organ Transfer Business Process Model

to generate code that copies the information into process variables. Equivalent snippets are assigned to the remaining tasks of the process model.

```
1 (bytes32 _bloodType, uint256 _age, uint256 _harvestingTime) : (bytes32 bloodType,
   uint256 age, uint256 harvestingTime) -> bloodType= _bloodType;
2 age = _age; _harvestingTime= harvestingTime; }
```

Listing 1: Snippet for the task “Provide donor’s health information”

5. Generation of smart contracts and execution of the organ transfer process

This section validates the feasibility of the proposed approach. We first explain the generation of smart contracts, and then we illustrate the creation and execution of a process instance in the Blockchain.

5.1. Generation of smart contracts

To generate “OrganTransfer” smart contract from a BPMN model, we use the “Compilation Tools” module of Caterpillar which establishes a comprehensive BPMN-to-Solidity mapping. Given a BPMN model in standard XML format, this module generates a smart contract in Solidity, which encapsulates the workflow routing logic of the process model. The proposed temporal constraint extension allows to Caterpillar to support, in addition to basic BPMN control flow elements (i.e. tasks and gateways), a large set of temporal constraints, such as duration, temporal dependency, temporal constraint over cardinality, and start/end temporal constraints. To generate smart contracts of the organ transfer case study, we need to perform the following steps.

- (i) Run Caterpillar using Glup, a toolkit that automates tasks in a development workflow (Figure 5a).
- (ii) Submit the BPMN model using a HTTP POST request on the URI “/models” with a JSON message that includes the model name and the model serialized in the BPMN 2.0 XML format (Figure 5c). Here, we use Postman a HTTP client that tests HTTP requests. Consequently, the smart contract generation and compilation will be triggered (Figure 5b).
- (iii) Enable the process to interact with external services. To do so, a corresponding smart contract/address must have been previously registered using a HTTP POST request on “/services”. This is required because the service’s smart contract address is compiled into the process’s smart contract.

Listing 2 shows an excerpt of the generated OrganTransfer smart contract implementing the duration temporal constraint. The generated smart contract of the proposed use case is available at [10].

```
1     uint marking = 1 ;
2     uint taskTimer_duration = 0 ;
3     function takeFlight(uint minTime, uint maxTime) returns (bool) {
4         uint localMarking = marking;
```



```

5      uint startTime = now;
6      if (localMarking & 2 != 2) return false;
7      flight();
8      uint endTime = now;
9      taskTimer_duration = calculate_duration(startTime, endTime, minTime, maxTime);
10     step(localMarking & uint (~2) | 4);
11     return true; }

```

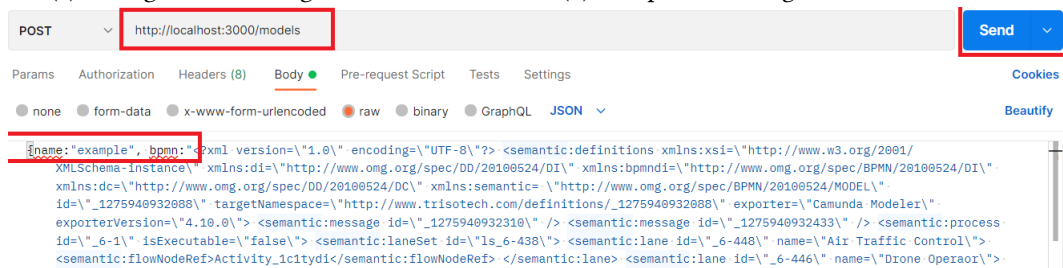
Listing 2: An excerpt of the generated OrganTransfer smart contract

```

C:\amal\Caterpillar\caterpillar-core> glup
...
Proxying: http://localhost:3000
Access URLs:
  Local: http://localhost:7001
  External: http://192.168.1.2:3000
-----
  UI: http://localhost:3002
  UI External: http://192.168.1.2:3002
-----
CONTRACTS
example :Process_1_OrganTransfer
example:Process_1_WorkList
PROCESSED SUCCESSFULLY
-----
POST /models 201 11725.158 ms
-----

```

(a) Configuration of Organ Transfer Process (b) Compilation and generation of smart contracts



(c) Submission of the BPMN model using a HTTP POST request

Figure 5: Steps of smart contracts' generation

5.2. Creation and execution of a process instance in the Blockchain

To execute a process instance of the “OrganTransfer” process in the Blockchain, we use Caterpillar’s Execution Engine. Through this Execution Engine, we can deploy the smart contract generated by the Compilation Tools. Consequently, we can create instances of the deployed process model, to identify enabled tasks and to control their execution. Meanwhile, executing script tasks and triggering external service calls are performed by the Execution Engine automatically. Besides, to create and track process instances graphically we can use Caterpillar’s Execution Panel. To execute a process instance of the organ transfer case study, we need to perform the following steps.

- (i) Create a new instance using a HTTP POST request on the URI `/models/:mid`, where `:mid` is the identifier rendered at the registration of the process model.
- (ii) As a response to this request, Caterpillar will return a hyperlink as a reference to the newly created process instance and “201” as a HTTP status. The returned hyperlink has the form `/processes/:pid`, where `:pid` represents the Ethereum address assigned to the instance of the process smart contract (Figure 6a).

- (iii) Use the execution panel to reproduce graphically the results of the state of a process instance and also to enable the execution of user tasks. Here, enabled tasks are displayed in dark green, while activities in execution are depicted in light green (Figure 6b).
- (iv) Execute an enabled task by clicking on it and submitting the required parameters, as illustrated in Figure 6b. Only user tasks and catching messages are depicted in the execution panel, because the execution of other types of BPMN elements (e.g. gateways, throwing events, script tasks, etc.) is performed internally by Caterpillar.



Figure 6: Steps of the execution of a process instance

6. Evaluation

In this section, we evaluate the proposed solution with respect to financial cost to assess its feasibility for use within a real-world setting.

Transactions on the Ethereum Blockchain induce a certain transaction fee. Ethereum uses a unit called gas to measure the amount of operations needed to perform a task, e.g., deploying a smart contract or executing an ABI. It is always essential to estimate the gas costs while developing a smart contract and so as to avoid extra charges. Fortunately, Caterpillar’s Compiler module generates optimized smart contract codes through its mapping rules. Indeed, mapping rules which include the marking variable, the step function, and the bit-wise operations (see section 2), address the gas consumption cost issue and generate minimized and optimized codes. The mapping rules of the proposed temporal constraints extension contains a time variable, a time function and a time guard to keep harmony with the mapping rules of the Caterpillar compiler, and thus remaining optimized for future expansion. Table 2 lists the transaction cost for some operations including deploying smart contracts, and executing some tasks such as “Provide donor’s health information”, “Send flight telemetry and environmental conditions”, and “Update the arrival time”

Table 2
Operations' Gas Cost

Operation	Gas
Smart Contract Deployment	1,350,248
Provide donor's health information	165,730
Send flight telemetry and environmental conditions	204,410
Update the arrival time	86,449

7. Related Work

Temporal constraints are a well studied topic in BPM [5]. However, prominent blockchain-based BPM systems such as Caterpillar [2], Lorikeet [3], and other work [4] do not mention temporal constraints at all. Some approaches are aware of time related issues and leave them for future work [1, 11]. A limited number of work take into account temporal constraints in blockchain-based BPM systems. Initial work have performed the simulation and prediction of temporal aspects in blockchain-based process execution. Yasaweerasinghelage et al. [12] use process simulation to send transactions to a private Blockchain in order to evaluate the confirmation time as well as latency. Haarmann et al [13] estimate, through manual simulation techniques, the total duration of choreography models in face of changing inclusion times and block times. However, both work assume that there is a readily available measure of time and ignore the real source of the timing information. Mavridou et al. [14] generate Solidity code from a finite-state machine based model while taking into consideration delayed processes. However, the latter work is still not mature enough and needs more development to cover a richer set of temporal constraints for business processes. Recently, Ladleif et al. [15] introduce a set of time measures available on Blockchain platforms such as block timestamp, block number, smart contract parameter, storage oracle request-response oracle. They also provide a systematic discussion and evaluation through important metrics like accuracy, trust, immediacy, cost, and reliability. Finally, our prior work [6] extend Caterpillar to support the modeling and execution of various temporal constraints for business processes such as duration, temporal constraint over cardinality, temporal dependency and start/end temporal constraints. The proposed implementation on Ethereum uses time-guards checking and block timestamps to evaluate these temporal constraints. This prior work is the first research attempting to consider BPs temporal constraints in Blockchain smart contract code. However, it lacks the validation of its feasibility and effectiveness. The objective of this paper is to provide implementation and technical details of a particular case study to our prior work to show its feasibility.

8. Conclusion

In this paper, we introduced a Blockchain-based case study of organ transfer by healthcare drone delivery, which is highly time sensitive and constrained by hard timing requirements. This case study was implemented using Caterpillar combined with our proposed extension that enables the transformation of a large set of temporal constraints from a business process model to a smart contract code. In future work, we plan to explore further techniques that enable time

measures in Blockchain platforms in order to accurately deal with temporal constraints. This would improve time management in critical systems as organ delivery drones.

Acknowledgments

This work was partially supported by the LABEX-TA project MeFoGL: "Méthodes Formelles pour le Génie Logiciel".

References

- [1] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, J. Mendling, Untrusted business process monitoring and execution using blockchain, in: International Conference on Business Process Management, volume 9850 of *LNCS*, Springer, 2016, pp. 329–347.
- [2] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, A. Ponomarev, Caterpillar: a business process execution engine on the ethereum blockchain, *Software: Practice and Experience* 49 (2019) 1162–1193.
- [3] A. B. Tran, Q. Lu, I. Weber, Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management., in: International Conference on Business Process Management (Dissertation/Demos/Industry), 2018, pp. 56–60.
- [4] L. García-Bañuelos, A. Ponomarev, M. Dumas, I. Weber, Optimized execution of business processes on blockchain, in: International Conference on Business Process Management, volume 10445 of *LNCS*, Springer, 2017, pp. 130–146.
- [5] S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, The temporal perspective in business process modeling: a survey and research challenges, *Service Oriented Computing and Applications* 9 (2015) 75–85.
- [6] A. Abid, S. Cheikhrouhou, M. Jmaiel, Modelling and executing time-aware processes in trustless blockchain environment, in: International Conference on Risks and Security of Internet and Systems, Springer, 2019, pp. 325–341.
- [7] T. Nation, 20 per cent of emergency patient deaths blamed on traffic jam delays, 2019. URL: nationthailand.com/national/30304268.
- [8] M. Rucker, The potential of drones providing health services, 2020. URL: verywellhealth.com/potential-of-drones-providing-healthservices-4018989.
- [9] S. Scutti, First drone delivery of a donated kidney ends with successful transplant. *cnn.*, 2019. URL: cnn.com/2019/05/01/health/drone-organ-transplant-bn-trnd/index.html.
- [10] SC, Organ transfer smart contract, 1. URL: github.com/amal-abid05/OrganTransfer.
- [11] J. Ladleif, M. Weske, I. Weber, Modeling and enforcing blockchain-based choreographies, in: International Conference on Business Process Management, Springer, 2019, pp. 69–85.
- [12] R. Yasaweerasinghelage, M. Staples, I. Weber, Predicting latency of blockchain-based systems using architectural modelling and simulation, in: International Conference on Software Architecture, IEEE, 2017, pp. 253–256.
- [13] S. Haarmann, Estimating the duration of blockchain-based business processes using simulation., in: Central European Workshop on Services, 2019, pp. 24–31.

- [14] A. Mavridou, A. Laszka, Designing secure ethereum smart contracts: A finite state machine based approach, arXiv preprint arXiv:1711.09327 (2017).
- [15] J. Ladleif, M. Weske, Time in blockchain-based process execution, in: International Enterprise Distributed Object Computing Conference, IEEE, 2020, pp. 217–226.