UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Fortunat Lufunda Mutunda

Timer Service for an Ethereum BPMN Engine

Matser's Thesis (30 ECTS)

Supervisor: Luciano Garcia-Bañuelos

Tartu 2017

Taimeri teenused Ethereumi BPMN mootorile

Lühikokkuvõte: Plokiahela tehnoloogia võimaldab pakkuda platvormi rakenduste käivitamiseks ilma keskse volitajata. Muud atribuudid nagu püsivus, andmete manipuleerimatus ja levitamine muudavad selle tehnoloogia atraktiivseks ärirakenduseks. Tartu Ülikooli tarkvaratehnika rühm on huvitatud, et selles kontekstis rakendada täitmismootor BPMN mudelile, mis jookseks Ethereumi plokiahela tehnoloogia peal. Antud magistritööga lisatakse platvormile tugi BPMN taimer-sündmuste jaoks piiratud mustrite alagrupis. Tulemuseks on loodud prototüüp, mis on testitud hoiustusel põhineva rakendusega, mille eesmärk on juhtida omandi lühiajalist rentimist.

Võtmesõnad:
Blockchain tehnoloogia, Business Process Management, Eutereum

CERCS:P170 - Computer science, numerical analysis, systems, control

Timer Service for an Ethereum BPMN Engine

Abstract: Blockchain technology comes with the promise of providing a platform for execution of applications with no central authority. Other attributes such as persistence, tampering free and distribution make of this technology an attractive solution for business applications. In that context, the group of software engineering at the University of Tartu is interested in implementing a execution engine for BPMN that runs on top of the Ethereum blockchain. This work contributes with the addition of the support of BPMN timer events to the platform, in a restricted subset of patterns. The approach has been implemented in a prototype and tested with an escrow-based application for management of short term rental of properties.

Keywords:
Blockchain Technology, Business Process Management, Ethereum

CERCS:P170 - Computer science, numerical analysis, systems, control

Contents

Table of Contents

List of Figures

1 Introduction

In the recent years, there has been efforts in solving the issue of centralized databases (i.e. One company owns the database, which can be hacked and cause repercussion on people), there is also the problem of trust between companies which always requires a central authority. Because it is not always for two parties to come to an agreement without one trying to violate the rules. These issues have been addressed recently by the so-called blockchain technologies[But13; DB17; Nak09], which are currently attracting the attention of both industry and academic communities. A blockchain is a highly replicated, append-only database, which provides guaranties for concerns such as reliability, tampering resistance and trust. A blockchain is in some way a reminiscence of a ledger, where every transaction is recorded permanently in way that none of the transaction records in the ledger can be modified. Tampering resistance makes of a ledger a convenient tool, for instance, in the context of accountancy.

Bitcoin [Nak09] is probably the most well-known example of a digital technology implementing the idea of a digital ledger or a blockchain. Bitcoin has not only shown that the use of cryptographic methods, replication and consensus make it possible to implement a digital ledger, but also that this technology can be a platform for performing safe exchange of cryptocurrencies. Along the same lines, in 2015 an open source project called Ethereum [But13] entered the arena of blockchain technologies by providing not only a simile of Bitcoin, but a more flexible platform by the introduction of the so-called smart contracts[Web+16], which have opened new opportunities for development on top of blockchain.

The Software Engineering Group at the University of Tartu has started efforts in using this platform with a lightweight BPMN execution engine. The efforts of this work has been focused on implementing the BPMN core constructs (cf. script tasks, exclusive and parallel gateways), by generating smart contracts that implement the process specified in a given BPMN model. However, the project is still in early stages and more development is required to cover a richer set of BPMN constructs.

It is in this context that my Masters' thesis work takes place. More specifically, my work aims at providing support to Timer Events as defined in the BPMN standard[WG;

DR13; LWR14; SH99]. Timer events are an interesting construct in BPMN because they provide ways to specify temporal constraints such as deadlines and/or milestones in the underlying business process[DR13; WG; LWR14; SH99]. However, the notion of time in a technology such as Ethereum does not have a clear meaning: there is not global clock, and the time for a transaction completion is not fixed (the same transaction can take a few seconds or several minutes). Nevertheless, we are aware of one existing solution that provides a sort of timer for Ethereum and, in discussion forums, it is also claimed that a timer service for Ethereum can implemented via an external service[Etha; Mer]. However, it is unclear if such solutions can be applied into, and what would the implications of using them in a BPMN execution engine. Those are in fact the main goals of my research.

The rest of this research is structured as follow, chapter 2 is about the literature review, background and related work. chapter 3 the design and implementation of the different types of timer event onto the execution engine. Next, chapter 4 is the case study analysis of the implementation and finally chapter 5 is the summary and future work.

2 Background

In the content of this chapter, we will evaluate the tools and pre-existing technologies. First, we are going to discuss Business Process Management, and we will discuss the timer event. Secondly, we are going to introduce Ethereum and Smart contracts. Moreover, finally, we will talk about the Ethereum alarm clock.

2.1 Business Process Management

Business process management (BPM)[OMG10; OMG11] is a discipline that provides an overview of an organization's business process and their interactions to optimize and automate them as much as possible. To achieve these changes, it is necessary to analyze the way the organization functions to visualize it by modeling it with BPMN (Business Process Model and Notation) and related tools. After modeling the process, the second step is often the process automation, sometimes partially, which are then monitored to identify the restriction to be undertaken [DR13; RLP16].

2.2 BPM Life cycle

The life cycle of a BPM initiative comprises 7 phases [DR13; OMG11]. It starts with the identification of the underlying process, and is then followed but process discovery, process analysis, process redesign, process implementation and finally process monitoring, and the cycle begins again (see Figure 1)[RLP16; DR13]. Each of these phases is described further in the following:

- Process identification: During the phase of identification we enumerate and prioritize the different business process in an organisation. We scope and define different relations between each of the business processes to finally have an outcome of a process portfolio which can give us relevant dimensions of the business process(.i.e if the process is healthy or not or what has a higher priority than the other in the business process); The process architecture(the organisation of the business process inventory accordingly)[DR13]

- Process discovery: This is the phase where we make the "*AS-IS*" process model by gathering information from different members of the organization etc. In another words, we discover the business process.
- Process analysis: In this phase we take the current "*AS-IS*" process and we do some analysis to find the ongoing issues with the process. A qualitative analysis(.i.e.the route cause analysis, we draw a why why diagram and to find why some issues are happening), a quantitative analysis for example a process simulation.
- Process redesign: After finding the weakness in the "*AS-IS*" process model we will redesign it into a "*TO-BE*" process which eliminate the issues of the "*AS-IS*" process model by making some trades e.g. time or performance.
- Process implementation: We try to put the process into execution, this include redesign the process by automating some of the process and probably hiring or training people that have to deal with some new change that have occured into the business process.
- Process monitoring: Consisting in analyzing the state of the processes through dashboards showing the performances of the processes, events logs etc.

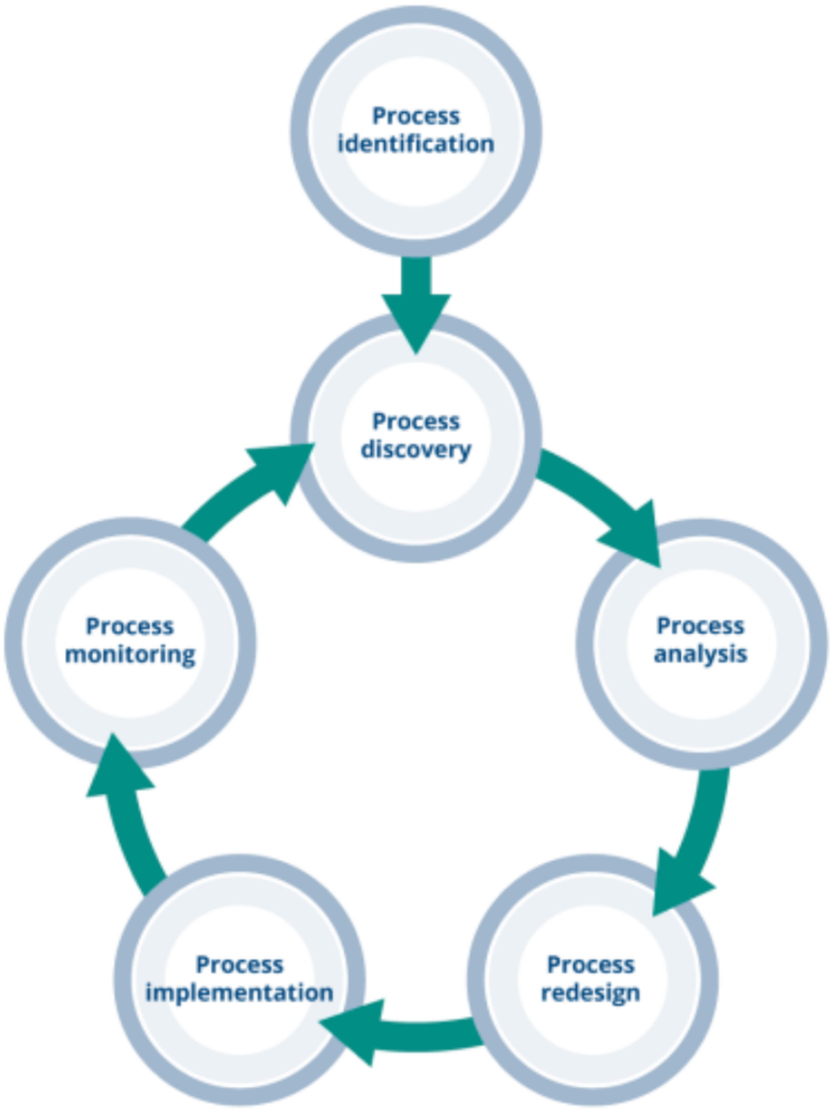


Figure 1. BPM Life Cycle [DR13]

2.3 Business Process Model and Notation

The main purpose of BPMN is to provide a notation that is understandable to all the users, from business analyst who create the initial draft of a process to the developers who will responsible of developing the technology that will execute the given process, and ultimately, to the users of the company that are going to implement these processes. Thus, BPMN creates a standardized bridge to fill the gap between business process modeling and the implementation of procedures.

BPMN represents an initiative to capitalize on and consolidate good practices in the various methods and notations that existed in the field of process modeling. BPMN and UML (Unified Modeling Language) are two modeling specifications developed by OMG (Object Management Group [1]) that are not competing but complementary. UML would be usually used for the analysis and design of information systems while BPMN is intended for the analysis and design business processes that involve and interact with the aforementioned systems. It is thus possible to change from a process diagram defining the business requirements in BPMN to UML case diagrams to document the requirements for the systems involved.

BMPN is based on three types of models[OMG10; DR13; OMG11; RLP16]:

- Process model to represent the flow of an organization's internal processes as well as public processes (i.e., interfacing with external third party activities [DR13; OMG10]).
- Models of collaboration to represent the processes of several entities and the exchanges allowing to link these processes[OMG10; Ora; Cama; RLP16].
- Choreography models to represent expected behaviors of actors in a process[OMG10; Ora; Cama; RLP16].

BPMN allows to represent internal business procedures, but also B2B systems[OMG10; F09], through public processes and choreographies, as well as advanced process orchestration concepts such as exception handling and transaction clearing.

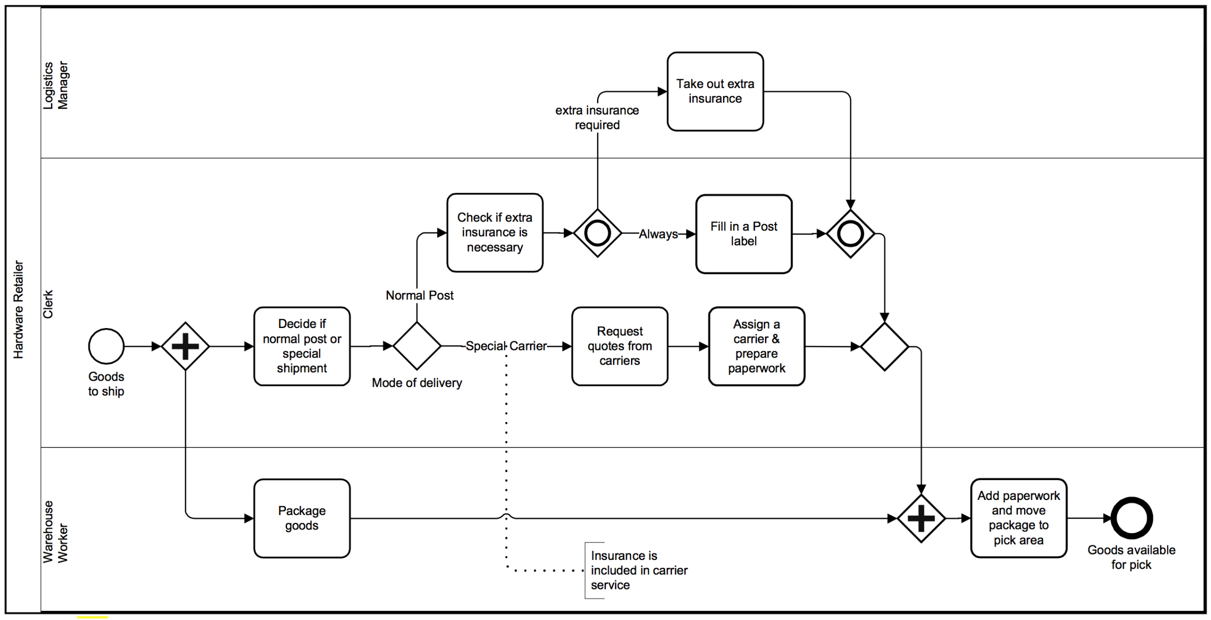
These models combine the following core elements(Figure 2):

- The *"pool"* and the *"lane"* represent the entities and participants in a process
- The *activities* ("activity") represent the decomposition of the process
- *Events* ("event")

[1]omg.org

- *Sequence flows* represent the sequence of these elements
- *Gateways* allow the splitting or convergence of control flows
- *Messages* and message flows are used to materialize exchanges
- *Data*
- *Annotations and associations between related elements*

Many variants of these basic elements and activity markers further clarify the meaning of the model[OMG11; Kos+14; DR13].

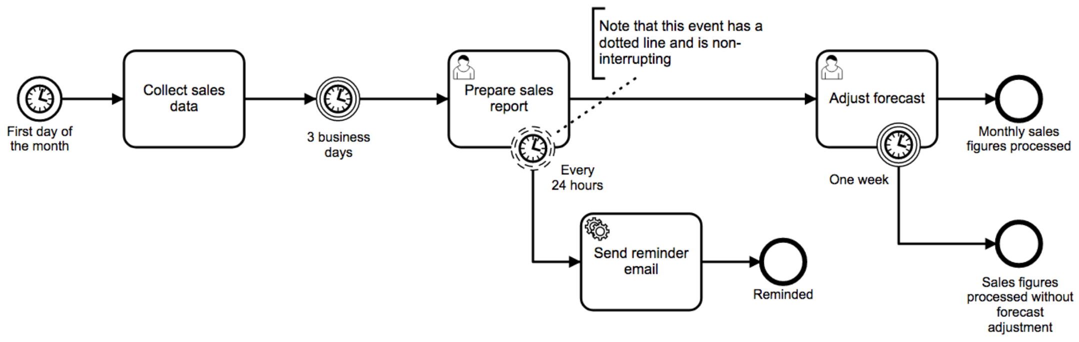


Hardware Retailer
Logistics Manager
Clerk
Warehouse Worker
Take out extra insurance
extra insurance required
Check if extra insurance is necessary
Always
Fill in a Post label
Normal Post
Goods to ship
Decide if normal post or special shipment
Mode of delivery
Special Carrier
Request quotes from carriers
Assign a carrier & prepare paperwork
Package goods
Insurance is included in carrier service
Add paperwork and move package to pick area
Goods available for pick

Figure 2. BPMN Core Concept [OMG10]

2.4 Timer event

The timer event, as it names says it is also a type of event that can indicate the start of a process, this means, a process can occur only after a specific temporal event, e.g. every day 12 o'clock, every 10th day of the month, or every Monday. But the timer event can also be used as an intermediate catching event, e.g. a deadline or timeout, to indicate



Note that this event has a dotted line and is non-interrupting
First day of the month
Collect sales data
3 business days
Prepare sales report
Every 24 hours
Send reminder email
Reminded
Adjust forecast
One week
Monthly sales figures processed
Sales figures processed without forecast adjustment

Figure 3. Different Types of timers events[Camb]

that a process can only proceed after a certain amount of time has happened between one activity and the previous one [DR13; Biz; OMG11; OMG10; Ora]. These particular traits make the timer event a catching event only this time just flows but itself and upon reaching a certain time then the process cannot start or continue, because time is outside of anyone's control therefore the event can only wait before triggering another.

BPMN provides four variants of timer event, with two of them being interrupting events and the other two being not-interrupting events. Usually, an interrupting event would abruptly interrupt the execution of an ongoing subprocess or task when a dealine is reached. An non-interrupting event, on the other hand, is used for triggering some background and/or parallel processing at a given time, without stopping the ongoing process. The following Figures illustrate both types of timer events.

2.5 Ethereum

Ethereum is a decentralized exchange protocol allowing the creation by users of smart contracts thanks to a Turing-complete language, see [But13]. These smart contracts are based on a computer protocol to verify or enforce a mutual aggreed outcome. Smart contracts are deployed on the blockchain where they can be publicly accesses (e.g. executed).

Ethereum uses a crypto-currency called *Ether* as the means of payment for these contracts. Its corresponding symbol, used by the trading platforms, is *"ETH."*. Ethereum is the third largest decentralized cryptographic currency with a capitalization of billion of euros.

2.5.1 Smart Contract and Execution cost

Smart Contracts are computer programs that can be deployed and ran on the blockchain. The Ethereum consortium has defined the following contract-oriented programming languages: *Serpent*[Ethb] or *Solidify*[Ethc]. Hereinafter, we will focus on Solidity which is the most widely use language so far. Listing 1 provides an example of a smart contract implemented in Solidity.

```
pragma solidity ^0.4.11;

contract SimpleAuction {

    address public beneficiary;
    uint public auctionStart;
    uint public biddingTime;
    address public highestBidder;
    uint public highestBid;
    mapping(address => uint) pendingReturns;
    bool ended;

    event HighestBidIncreased(address bidder, uint amount);
    event AuctionEnded(address winner, uint amount);

    function SimpleAuction(
        uint _biddingTime,
        address _beneficiary
    ) {
        beneficiary = _beneficiary;
        auctionStart = now;
        biddingTime = _biddingTime;
    }
    ...
}
```

Listing 1. Smart Contract example [But13]

A smart contract in Solidity allows one to define a set of state variables (lines 5-11), whose value is made implicitly persistent after every transaction, a set of contract events (lines ...), by means of which the contract communicates facts with the external environment and a set of functions (lines ...). It is by means of functions that the user specifies the business logic underlying a smart contract. Generally speaking, Solidity provides two types of functions: functions that change the contract state and functions that can be used to query the current state of the contract.

The execution of a smart contract, whether it is a simple transfer of Ether between two accounts or the execution of several lines of the code of a contract, requires paying the minors for the tasks' execution[But13]. This remuneration is done in Ether on an infinitesimal scale and is then called *gas*. Each operation on Ethereum costs the gas that corresponds to the effort to be provided to process it. The price of gas varies according to the market: each miner can set his price and corresponds to the number of Ether he/she wishes to receive for the effort he provides.

In June 2016[But13; Ethc], the average gas price was 0.0000000225 Ether25. Thus, a basic transaction of transfer between two addresses requiring 21000 gas corresponds to an average cost of 0.00047 Ether in processing costs. To avoid certain contracts becoming overpriced when the price of Ether has increased; In fact, the number of gas required for execution is defined by the complexity of the operations while the price of the gas can be adjusted according to the course of the Ether; To prevent an infinite loop in a code from turning forever because at the time when all the gas supplied in the transaction was consumed, the minor stops processing the transaction and records it as is. The user chooses the price he/she is willing to pay: if he pays the average price, the execution of his contract will take much longer since all the more profitable transactions are executed in priority.

2.6 Ehtereum Alarm Clock

As mentioned earlier about the way time works on Ethereum, there are two more things that we need to explain if we want to grasp what the problem Ethereum alarm clock is solving[But13; Etha; Mer]. Each Ethereum user has an account, and this account has a private key. However, contracts do not have a private key. Since the private key based accounts are those operated by humans; Smart contracts accounts are deployed computer capable of executing a program. However, all the code on the EVM must be trigger by a private key based account by sending a transaction which will execute something. The other issue is the fact that as soon as we send a transaction, this transaction is automatically executed as soon as it is included in a block; The design of Ethereum's protocol does not present any way of letting a transaction to be executed later.

When a transaction is created, one contract will be created holding the information of the transaction request(Figure 4). When it is time to execute the transaction, another contract sends the transaction request to execute that contract. Finally when all is done, then the account that requests the transaction in the first place will pay the transaction gas and the instantiation gas to both. Note that the transaction happens in a time window, not at a given moment, see[Etha; Mer].

```
contract SchedulerInterface {
    //
    // params:
    // - uintArgs[0] callGas
    // - uintArgs[1] callValue
    // - uintArgs[2] windowStart
    // - uint8 windowSize
    // - bytes callData
    // - address toAddress
    //
    function scheduleTransaction(address toAddress,
                                 bytes callData,
                                 uint8 windowSize,
                                 uint[3] uintArgs) public returns (address);
}
```

Figure 4. Transaction Scheduler[Etha]

3 Implementation

The stages used in the approach are depicted in Figure 5. The first step of the process takes a BPMN file as input, which is then parsed. As the programming language used in the implementation is Typescript, the system uses a library (i.e. bpmn-moddle[2]) to read the XML file and create an in-memory representation of the process model using Typescript objects. In the second step, the process model is analyzed (using the in-memory representation) to determine the underlying control flow information (i.e. list of nodes and edges in the model). Finally, the Solidity code is generated in the third step by using the control flow information and the in-memory representation of the process model, which were computed in the previous steps.

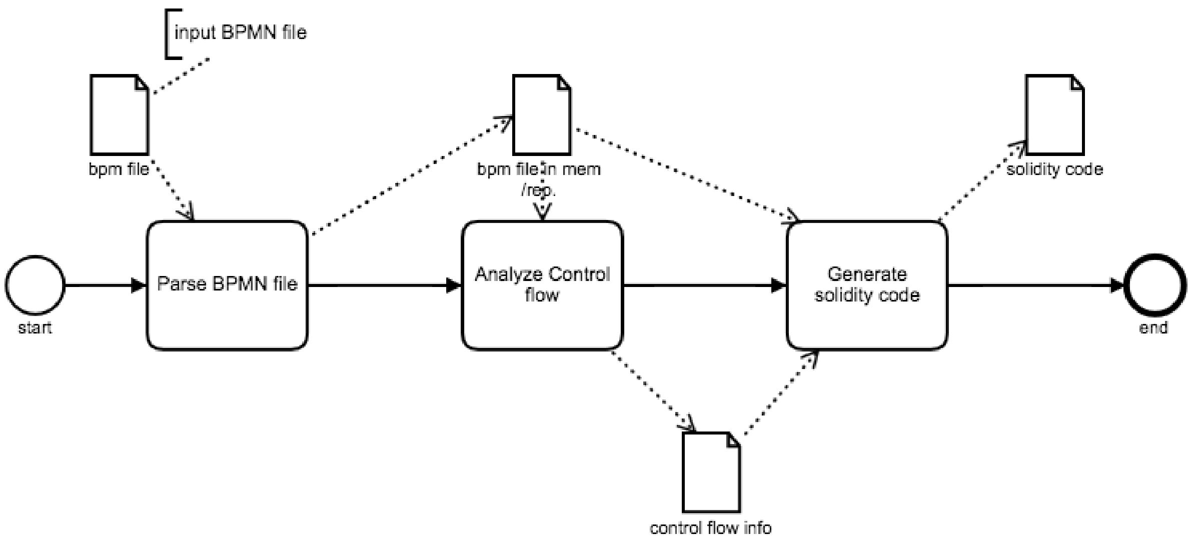


Figure 5. Implementation Process

3.1 Code generation

In this section, we will describe how an input BPMN process model is translated into Solidity code. Generally speaking, the dynamics of the execution of a process is implemented in a way that resembles the "token game" that is commonly used with Petri

[2]https://github.com/bpmn-io/bpmn-moddle

nets. The idea is the following, when a process model is instantiated, a token is put in the flow edges outgoing the start event. An activity gets enabled whenever a token is present in one of its incoming flow edges. When an activity is executed, the token in the incoming flow edge is removed and a new token is added into (every) outgoing flow edge. Note, that we assume that one flow edge will hold at most one token. The latter is a common assumption in the literature and is referred to as one-safeness. Based on this assumption, we will represent the execution state of a process instance (aka case) with a bit array: each flow edge is associated with a slot in the array and the value of such slot is identified with the presence/absence of a token in corresponding flow edge. Moreover, the full state of a process model, that is the overall distribution of tokens, will be referred to as the marking.

Let us illustrate the intuition with the example in Figure 6 and its corresponding Solidity code, which is shown in Listing 2. In the example, we have use two different types of activities, namely user tasks (tasks decorated with a person-like icon) and script tasks (decorated with a paper-like icon). According to the BPMN specification, user tasks must be used when an external actor, basically a user, interacts with the business process. In the context of blockchain, this could corresponds to tasks where users agree to pay some ether or to enter information for the processing of the business process. On the other hand, script tasks are used to represent processing steps that can be executed internally by the blockchain, for instance, to enforce a business rule (e.g. checking process data or the balance in ether for a given user). It should be clear, that the execution of a user task requires interaction and for this reason is implemented as an externally visible function in Solidity. On the other hand, a script task is implemented as an internal function and, hence, can only be called from the Solidity contract.

For convenience, each edge flow in the model is labeled with a red number, which corresponds to the slot in the bit array. Solidity does not provide an explicit data type for bit arrays, but we use integer values to this end. More specifically, each smart contract has a variable `tokens` (line 3). As we mentioned above, after instantiation, a token will be put in the flow edge that outgoings the start event. Since such flow edge is labelled as edge 0, the initial marking will be encoded as a 1 in the position 0 of the integer, i.e. tokens is 2^0.

Under the circumstances described above, task A will be enabled. In Solidity, that means that the execution of the function A in the smart contract (lines 5-11) will lead to a positive outcome, i.e. if called it would return `true`. It is worth noting that the Solidity code checks in line 7 if a token is actually present in position 0 of the integer `tokens`. Now, if task A is executed the token in position 0 needs to be removed and a new token would be put in the edge flow that corresponds to position 1. The latter can be implemented with bitwise operations: `tokens & uint( 1)` corresponds to removing the token from position 0, and `tokens | 2` corresponds to adding a token in position 1 (i.e. 2^1). Since read/write operations on state variables has cost in ether, we update not the variable `tokens` but a local copy of it `localTokens`.

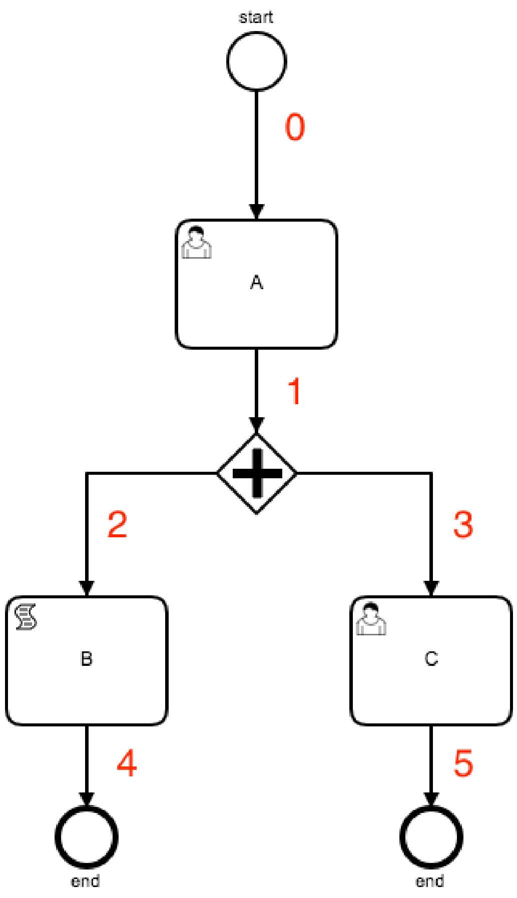


Figure 6. Simple sample BPMN model (no timer event)

1

```
contract Example_Contract {
    uint tokens = 1;

    function A() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 1 != 1)
            return false;
        step( localTokens & uint(~1) | 2);
        return true;
    }

    function B(uint localToken) internal returns (uint) {
        uint localTokens = tokens;
       // code to be executed
        return localTokens & uint(~4) | 16;
    }
    function C() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 8 != 8)
            return false;
        step( localTokens & uint(~8) | 32 );
        return true;
    }
    function step(uint localTokens) internal {
        bool done = false;
        while (!done) {
            if (localTokens & 2 == 2) {
                localTokens = localTokens & uint(~4) | 12;
                continue;
            }
            if(localTokens & 4 == 4){
                localTokens = B(localTokens);
                continue;
            }
            if (localTokens & 16 == 16) {
                localTokens = localTokens & uint(~16);
                continue;
            }
            if (localTokens & 32 == 32) {
```

```
41                  localTokens = localTokens & uint(~32);
42                  continue;
43              }
44              done = true;
45          }
46          tokens = localTokens;
47      }
48  }
```

Listing 2. Simple process smart contract

As mentioned earlier we have different tasks which could correspond to different types of activities. Now, the executions of these tasks also depend on whether their are automated or require external interaction. Let us consider the example in Figure 6 on which we can see two different types of tasks in action. Task A is a user task, that means after the user has done what they were supposed to do(i.e. fill a form and validated). Then the token will more from position 0 to position 1 where there is a parallel gateway. Some tasks and all gateways are automated, in such a way that when token reaches the position the process are implement as internal functions(see Listing 2, lines(13-17)) and are self-executed then the token will move to the next position. Hence, once the token his on position 1, the and-split gateway will be executed then one token will be placed on position 2 and 3. As task B on position 2 is a script task, hence automated it will run and move to token to position 4 then finish its side of the process. On the other hand, there is still a token on position 3, taking into consideration that task c is a user task, then the process would not execute directly but wait for user interaction. Right after the users have done what they had to do then execute the task then the process will continue and end. That is the reason why there function `step`(see Listing 2 lines 25-47) to manage all the cases mentioned. As its name suggests, the `step` serves to take steps during a whole business process execution by executing events(which are internal and do not require direct human interaction) and some of internal tasks that have to be self-executed.

3.2 Timer Event code generation

The EVM(Ethereum Virtual machine) does not provide the possibility of scheduling, that simply means it is not possible to just set a timer and expect it to be triggered at a given time. In the light of this document we have covered two cases of intermediate timer event. The first one being the case when a timer is in between two tasks and acts a stopwatch, and the second is when the timer is one of the deferred choices right after an event-based gateway. Here is the link to the project repository in Bitbucket[3].

3.2.1 Delay

The way we implemented the timer event remains true to the token game just with a small addition, a guard to check if a task is preceded by a timer or not. We are going to give a comparative example of a normal task and a task preceded by a intermediate timer event. On Figure 7 we have the example of a simple process, with a start event, task A, task B and the end event. As we are using the "token game" to go through each step, the flow of this process will be as expected: A token is place on position A, we then execute A. Afterwards we remove that token from position A to move it to position B. This procedure is repeated dependent on the size of a business process, however in our case the process execution will end after B.

Each node contains a guard that checks if the token is at a given node. After checking, the node will execute and then the token is removed and pass the next node. Therefore, at this point we only need one guard(see Listing 3, line 13) which does check where is the token and see if the token is still on position B or it should move to the next direct successor. And if B has a successor, this successor will do the same.

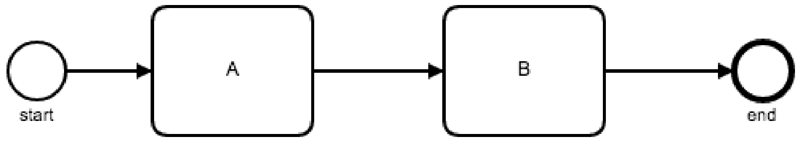


Figure 7. Process with no delay

[3]https://bitbucket.org/lgbanuelos/bpsol-frontend/branch/fortunate

```
contract MyContract{

    function A() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 2 != 2)
            return false;
        step( localTokens & uint(~2) | 4 );
        return true;
    }

    function B() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 4 != 4)
            return false;
        step( localTokens & uint(~4) | 8 );
        return true;
    }
    // code here...
}
```

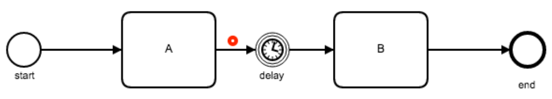
Listing 3. Smart contract with no delay

In contrast to that, there are cases when a task has to be delayed for some reason. That implies the setting of time which will mean the green-light of the task execution. i.e. In cases when a person purchases an article at the store, and they are given 14 days to try it out, withing those 14 days they can return it and get the full refund but after that period they can not. That is one the cases when we need an intermediate timer event to handle such delayed times. In this case, the procedure is a bit similar to the previous with differences. Since we now have a timer one guard will not suffice. Therefore we had an extra guard to meet the timer requirement. The procedure is a follow:

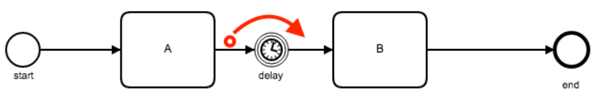
- First of all, if a process contains a timer event, at the creation of the contract we initialize a variable with the same name as the timer event with and ending with the expression “time”(see Listing 4, line 3). This variable will later on hold the value of the timestamp when the timer is called.
- Secondly, the timer function itself is internal,that is to say once the token reaches

the timer event, the function will take a timestamp(see Listing 4, lines 7-10) then remove the token and place it in the position of its following activity, see Figures 8b and 8c.

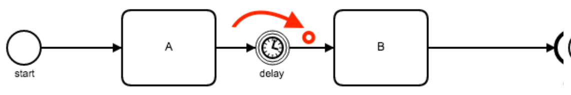
- Thirdly, as compared to the previous example with no timer, the difference is not big but it is major. We have added an extra guard(a timer guard) that basically checks if the timestamp now is is higher comparing to the timestamp taken previously plus the time specified by the timer(see Listing 4, lines 16-17).



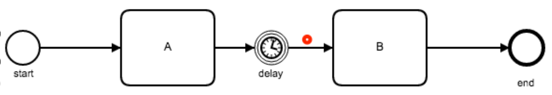
(a) step 1



(b) step 2



(c) step 3



(d) step 4

Figure 8. Timer delay

```
1  contract MyContract {
2      uint tokens = 1;
3      uint delay_time = 0;
4  
5      function A() returns (bool) {
6          uint localTokens = tokens;
7          if (localTokens & 2 != 2)
8              return false;
9          step( localTokens & uint(~2) | 4 );
10         return true;
11     }
12 
13     function delay(uint localTokens) internal returns (uint) {
14         delay_time = now;
15         return localTokens & uint(~4) | 8;
16     }
17 
18     function B() returns (bool) {
19         uint localTokens = tokens;
20         if (localTokens & 8 != 8)
```

```
            return false;
        if (now < 20 seconds + delay_time )
            return false;
        step( localTokens & uint(~8) | 16 );
        return true;
    }

    //more code below
}
```

Listing 4. Smart contract with delay

Note that `now` is an inbuilt Ethereum function, it is an alias to `block.timestamp`, which take the timestamp in UNIX time at the time at which it is called within a smart contract.

3.2.2 Timer on deferred choice

Event based gateways are not depending on data but on the event that occur first, i.e. A person has booked a hotel room and he or she is supposed to show up at a certain time or the room will be given to another person or if the person delays but call the hotel manager the room will be kept and finally if the person shows up on time they get the key and go to the room booked. This case shows the necessity of have a timer set which can be used as a last resort in a process(or decision making).

A depiction of a case where a event based gateway and a timer event are needed is show in Figure 9. It is worth knowing that in a case where there is an eventbased gateway once one event happens the others will be automatically cancelled, leaving only one path.

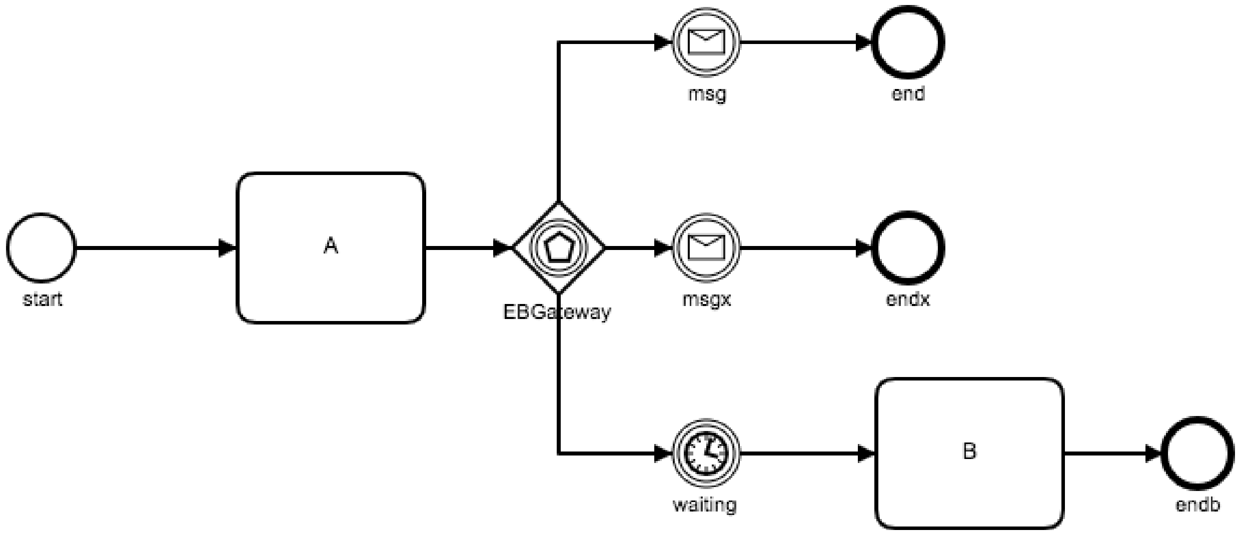
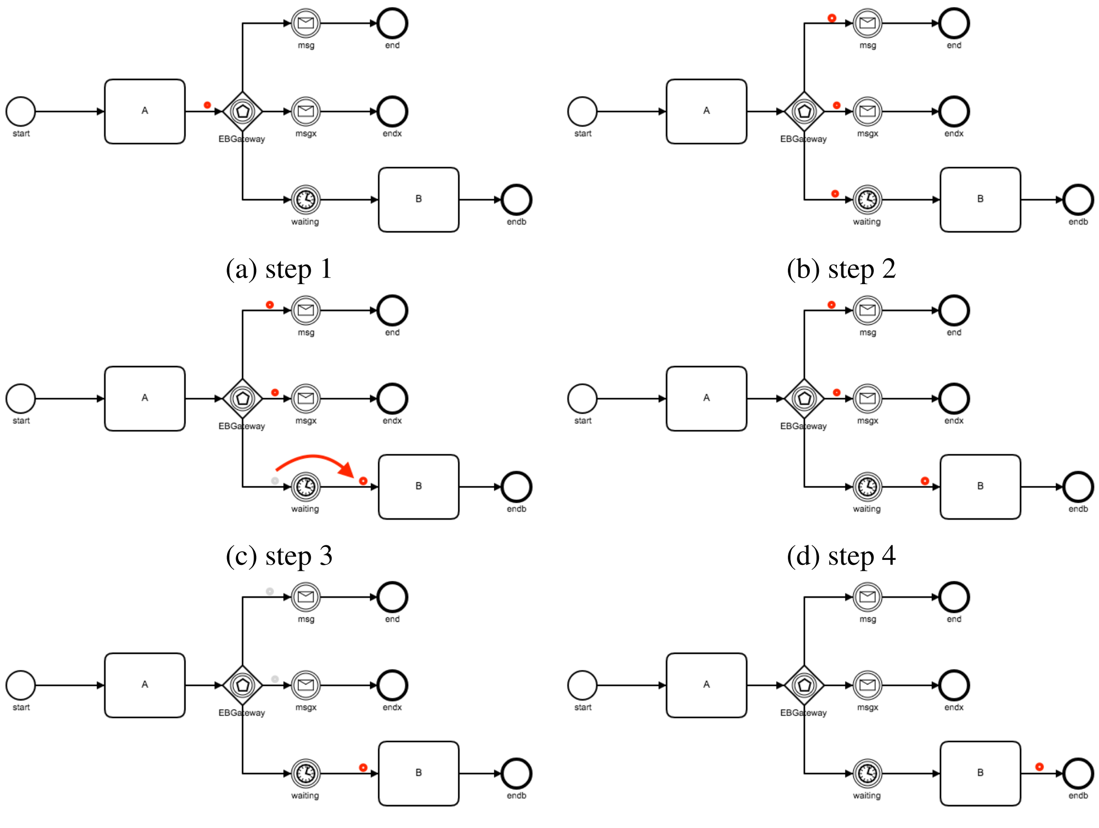


Figure 9. Deferred Choice

As shown in Figure 9 right after the event based gateway there are three options that can happen based on the choices that the persons involved can or will make, if none of the the first two options do not happen then automatically the last option will happen based on the timer has set time. We are going to describe step by step the means used to achieve this as represented on Figures 10.

As other processes, a token moves from one edge to another only then a node(task/event) can be executed.This rule remains even in the case of deferred choice, when a token is at the edge of an event based gateway(the gateway is executed internally), Figure 8a, the execution of the gateway distributes the tokens at each edge of its successors. This distribution enables the activation of each node at any moment all depending on the user's choice. As indicated on Figure 10b each intermediate catching event receives one token.

Note that on Figure 10b not all the intermediate catching events are just simple message events but one of the is a timer event. As mention in the previous section we only need to take a timestamp with the timer event and then place the token to the next edge for it to be executed at the appointed time. Therefore, when a token is put on the timer event, the system will execute it internally and take a timestamp then move the token to the next node, see Figure 10c and Figure 10d. As described in Listing 5 on lines 32-35 the timer will behave as it should then pass the token to the next node; the step function Listing 5 (lines 84-86), will automatically execute the timer.



(a) step 1 (b) step 2

(c) step 3 (d) step 4

(e) step 5 (f) step 6

Figure 10. Timer on deferred choice

```

contract MyContract {

   uint tokens = 1;

   uint waiting_time = 0;


    function msg() returns (bool) {

        uint localTokens = tokens;

        if (localTokens & 8 != 8)
            return false;
        if ( now >= waiting_time + 3 seconds )
           return false;
        step( localTokens & uint(~56) | 128 );
        return true;
    }

    function msgx() returns (bool) {
        uint localTokens = tokens;

        if (localTokens & 16 != 16)
            return false;
        if ( now >= waiting_time + 3 seconds )
           return false;
        step( localTokens & uint(~56) | 256 );
        return true;
    }

    function waiting(uint localTokens) internal returns (uint) {
        waiting_time = now;
        return localTokens & uint(~32) | 64;
    }

    function B() returns (bool) {
        uint localTokens = tokens;

        if (localTokens & 64 != 64)
            return false;
        if (now < waiting_time + 3 seconds)
            return false;
        step( localTokens & uint(~64) | 0 );
        return true;
    }

    function end() returns (bool) {
        uint localTokens = tokens;

        if (localTokens & 128 != 128)
```

```
            return false;
        step( localTokens & uint(~128) | 0 );
        return true;
    }

    function endx() returns (bool) {
        uint localTokens = tokens;

        if (localTokens & 256 != 256)
            return false;
        step( localTokens & uint(~256) | 0 );
        return true;
    }

    function endb() returns (bool) {
        uint localTokens = tokens;

        if (localTokens & 512 != 512)
            return false;
        step( localTokens & uint(~512) | 0 );
        return true;
    }

    function step(uint localTokens) internal {
        bool done = false;
        while (!done) {

            if(localTokens & 4 != 4){
                localTokekns = localTokens & uint(~4) | 56;
                continue;
            }

            if (localTokens & 32 != 0) {
                localTokens = waiting(localTokens);
                continue;
            }
            done = true;
        }
        tokens = localTokens;
        }
}
```

Listing 5. Smart contract with delay

Also, all the intermediate catching events part of a deferred choice aside the timer will also get a conditional guard. In that sense, if the specified time has reached they can no longer be activated. As described on line 26 in Listing 5(`if ( now >= waiting_time+ 3 seconds )`). Given this example, if the time specified by the timer(`3 seconds`) at which it should be executed. Added to the timestamp (`wating_time`) taken in the timer execution is less than or equal to *now*(the time at which we are trying to execute an event) then this event can not be executed, but they time alone.

4 Case Study

A project suggested at our University during the blockchain seminar was to implement a blockchain-based marketplace that enables people to list and/or rent short-term properties. The idea is comparable to services as such as AibBnB [4], HomeAway [5], etc. The main difference with other solutions is that there is no middleman, which potentially reduces some of the costs the costs for both guests and hosts. The suggested project is presented in the following chapter, which is then by discussion and potential future improvements.

4.1 ChainURent Project description

The system is built in such a way that only the portion of the application that requires currency transactions is deployed on the blockchain but the rest is stored in a regular database, considering that every transaction on the blockchain costs a fee. The system provides a web application/interface that allows a host to list all rooms/houses that is available for renting, with respect to specific criteria(i.e. the location of the place, the area in m^2, number of beds, kitchen, WI-FI, parking space, if it is possible to smoke, availability dates, etc.). A basic query interface is provided for the guest, using this interface, the guest enters the name of a city and a rental period of time. The system then return a list of available properties, matching the query input data (A more advanced query can be implemented that can allow a client to make more advanced queries, but since this does not have an impact on the blockchain; it is not the main focus here). Once the guest finds a property that meets his/her requirements, the system would allow this renter to book the property. We can consider that rent prices would be determined by the market (e.g. supply/demand).

[4]https://www.airbnb.com/

[5]https://www.homeaway.co.uk/

4.2 ChainURent Architecture

The core of the system is a smart contract implementing an escrow-based transaction. That is an understatement of the fact that there is no more need of middle man but the system will be the middle-man. Therefore, in order to book a property for a given period of time, both then host (owner) and the guest (renter) must deposit an amount of cryptocurrency that is equal to 50 percent of the daily rental price. Then the smart contract would keep the money from both of them. Since nobody loves losing money, the fact of keeping money from both client and owner increases trust between each other and removes the necessity of a middle-man whom would have been needed to keep not only their deposit money but also get paid for doing so. Booking cancellation will be possible up to 72 hours prior to the rental. Note that both parties (guest/host) are allowed to cancel a booking. The canceling party, however, will be penalized with the withdrawn of half of the deposit which will be given to his/her counterparts. Thus, the security deposit serves to protect one party from the other party violation on the terms of the rental agreement (e.g. the guest never shows up at the check-in date, the property was occupied, etc.).

Assuming that the property has a digital lock that can be opened with a 4-digit code. The code is sent by SMS on the check-in date when the guest transfers the full amount of the rent to the host via the smart contract. The 4-digit code will become invalid at the end of the rental period. A rating system is also available to allow the users(guest/host) to rate each other other and leave some comments.

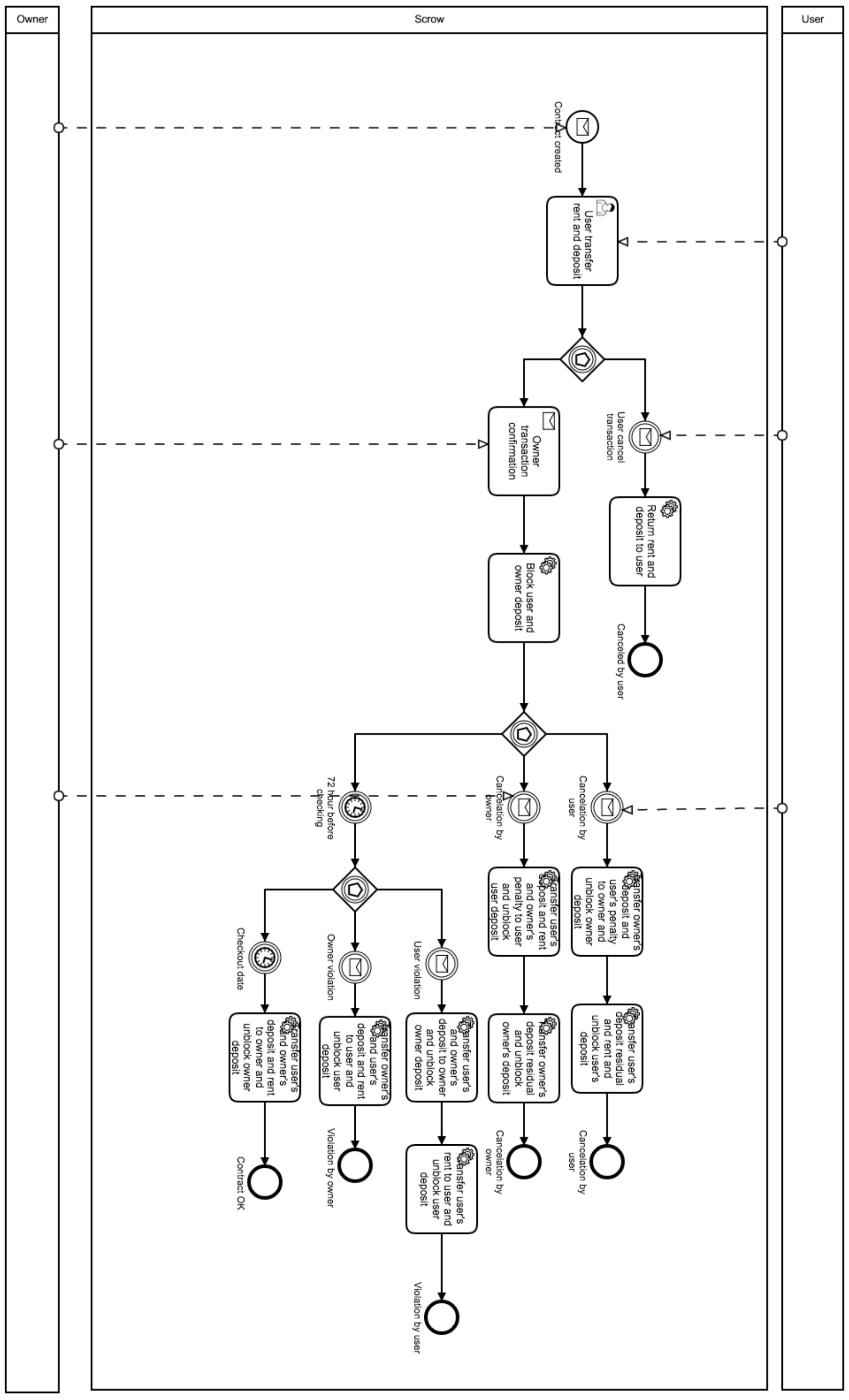

Owner
Scrow
User
Contract created
User transfer rent and deposit
User cancel transaction
Owner transaction confirmation
Return rent and deposit to user
Canceled by user
Block user and owner deposit
Cancelation by user
Cancelation by owner
72 Hour before cpecking
Transfer owner's deposit and user's penalty to owner and unblock owner deposit
Transfer user's deposit and rent and owner's penalty to user and unblock user deposit
User violation
Owner violation
Checkout date
Transfer user's deposit residual and rent and unblock user's deposit
Transfer owner's deposit residual and unblock owner's deposit
Transfer user's and owner's deposit to owner and unblock owner deposit
Transfer owner's and user's deposit and rent to user and unblock user deposit
Transfer user's and owner's deposit and rent to owner and unblock owner deposit
Transfer user's rent to user and unblock user deposit
Cancelation by user
Cancelation by owner
Violation by user
Violation by owner
Contract OK

Figure 11. ChainURent Business process

4.2.1 Timer Event ChainURent

The escrow system blocks both money from the guest and the client, so most of the time this technique will enforce people to always manage to be on time and respect the regulations of the platform. Obviously the happy path is the one when both the client nor the host suddenly cancel the booking and everybody get satisfied. However, if the user has to cancel before the appointed time the system will take half of the deposit money then give it to the host and the return also the deposit money from the host; If the owner happens to also cancel, the system will do the same on behalf on the user and then the process will end. Basically both the client and the host still have time to cancel before 72 hours before the time of check in, In case they do not the process will automatically move to the next step. When the time has reached, the time for check in has arrived, naturally at that point it is impossible to block "cancel" in that sense of the word but the process can still be terminated in three ways(see Figures 11,12).

First of if there is no issue or whatsoever during the entire stay, and the timer to checkout has reached, the only path that would be active is the one that allows the guest to take his/money because the timer-event will execute automatically and block the other two paths, the paths that allows one of the parties fire a violation action where their deposit will be sent back to them hence allowing them to cheat, buy staying in the premises and keeping the money on top(see Figure 13).

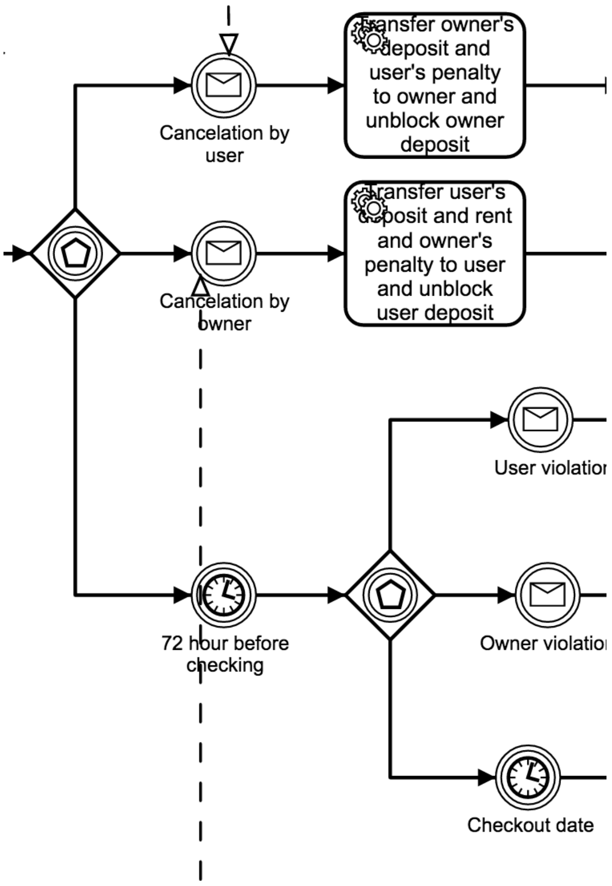
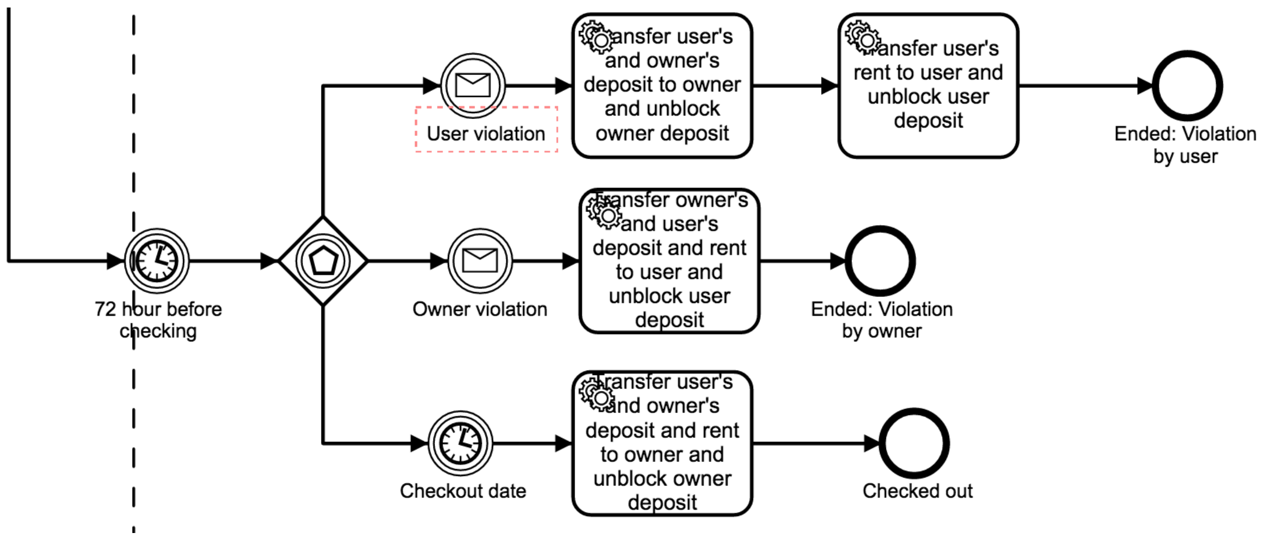

Transfer owner's deposit and user's penalty to owner and unblock owner deposit
Cancelation by user
Transfer user's deposit and rent and owner's penalty to user and unblock user deposit
Cancelation by owner
User violatio
72 hour before checking
Owner violatio
Checkout date

Figure 12. 72 hours before check in

Secondly, in case the time for checkout has not reached(the guest is still using renting the place), one of the two possibilities is that something occurs that was caused by the renter(guest) and that causes the the contract to be interrupted. Note this occurs during the stay not after, therefore since the smart contract timer has not registered the timestamp for checkout the host has the right to kick them out and end the contract. In other cases if the guest is not satisfied with the stay or if the place does not look like it was shown on the pictures or anything, he/she can also decide to break the contract, which is an understatement of owner violation, and just leave. In this later case the system will send back the right amount to the user, penalise the host and the will be the end of the business process.



Transfer user's and owner's deposit to owner and unblock owner deposit
Transfer user's rent to user and unblock user deposit
User violation
Ended: Violation by user
Transfer owner's and user's deposit and rent to user and unblock user deposit
72 hour before checking
Owner violation
Ended: Violation by owner
Transfer user's and owner's deposit and rent to owner and unblock owner deposit
Checkout date
Checked out

Figure 13. During stay

4.2.2 Deployment and Transaction Cost

In Bitcoin, an address has as its attribute a value corresponding to an amount used in a monetary system. In Ethereum, an address is associated with account objects. In the case of the user account, one finds the private key and an amount. In the case of the contract account, these two first attributes are found plus the hash of the code and the root of the data tree. The state of the Ethereum protocol is therefore the set of these attributes.

Each transaction specifies a destination address. If this address is a user, then funds are transferred in ETH (similar to Bitcoin or other cryptocurrencies). If this address is a contract, then the code runs in 3 ways:

- Send ETHs to other contracts
- Read or write to the associated memory
- Initiate the execution of other contracts

It is necessary to understand that all nodes in the Ethereum network run all of the transmitted transactions by running the code and then maintaining the total state of the protocol after execution. Thus, each of the nodes possess the same state of the protocol at a given instant. So we can see Ethereum as a set of programs on a single supercomputer that is massively decentralized, relocated and secured by thousands of machines.

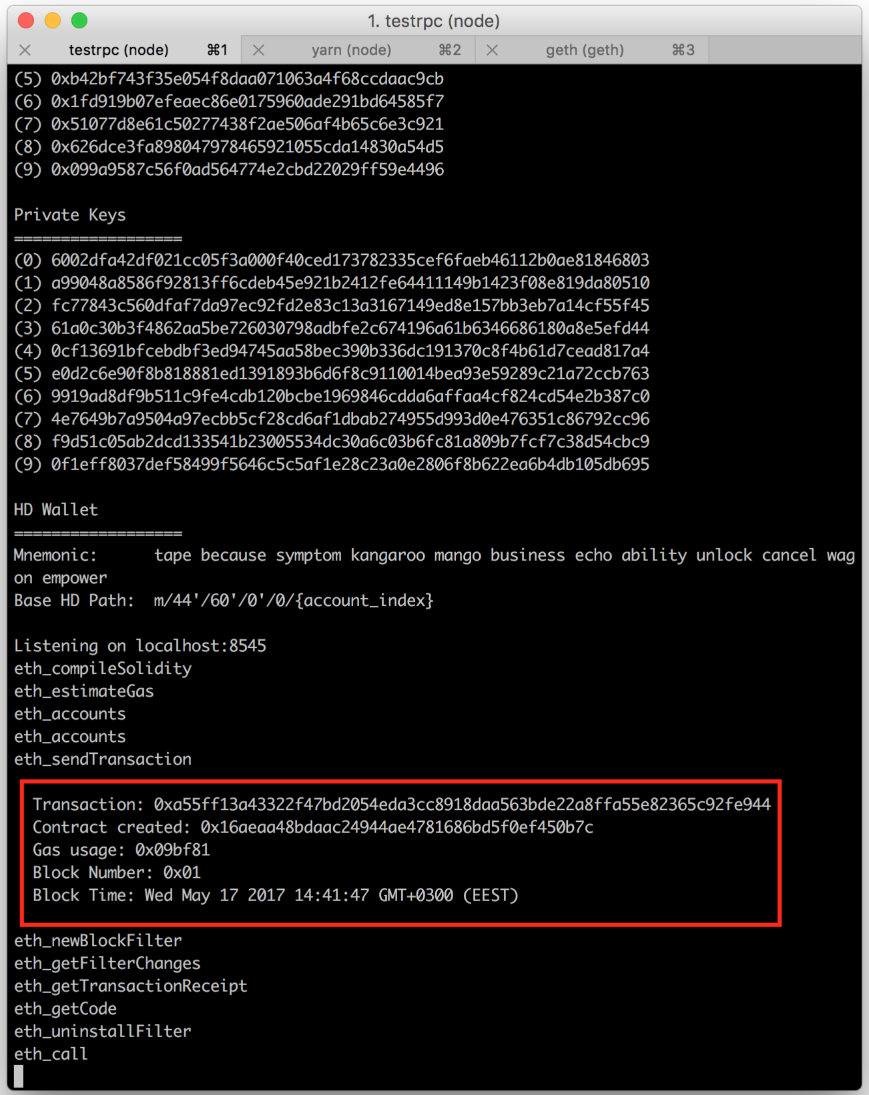
Ethereum solves the problem of shutdown where programs run indefinitely by charging each step of the code execution calculation gas. The blocks of the Ethereum blockchain have a fuel limit. Each transaction(see Figure 15) indicates the maximum amount of fuel and the price in Ether it wants to pay per unit of gas. If the transaction does not have enough gas, the performance is canceled, but the amount is still paid. The gas principle is an incentive to use network resources in a responsible manner. The gas limit, equivalent to the block size limit in Bitcoin, can be changed by a simple voting system when a block is generated, but other strategies can be considered.

Ethereum transactions contain the following:

- A nonce (an arbitrary number used only once) to avoid replay attacks
- Gasprice (amount of Ether per unit of fuel)
- The startgas (maximum amount of usable fuel)
- The destination address
- Amount in `ETH sent`
- Data (readable by the contract code - adds fuel costs)
- v, r, s (data values for the cryptography of elliptic curves)

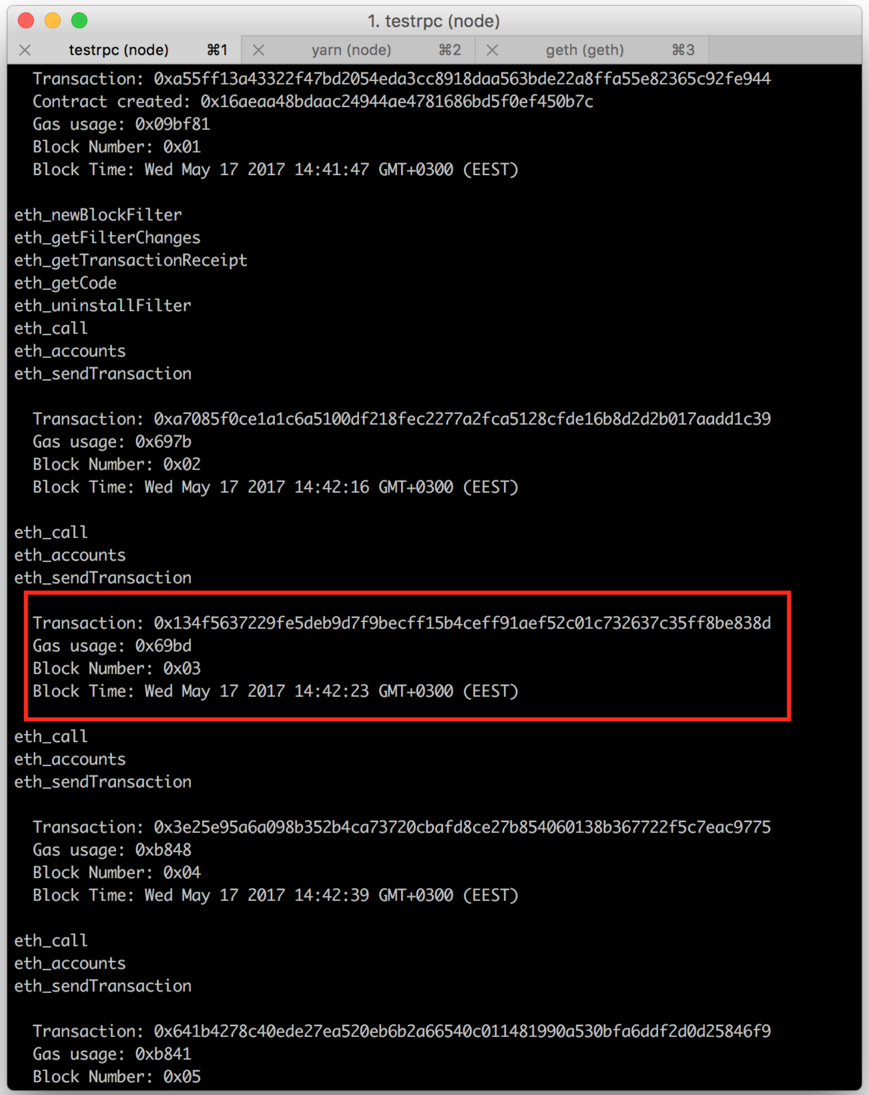
When a contract is deployed or a transaction is completed, a receipt object is created and hashed in the blockchain to validate and count transactions. It shows the intermediate state of the protocol after execution of the contract, the amount of fuel used and the logs(see Figure 15, 14). These are only valid in the current block and are not retained later. They are therefore not accessible by contracts, but are used by a thin client to view events and transaction history.

Now concerning *chaiurent*, since it is the owner who deploys the contract on ethereum then it is not be profitable it a contract had to be deployed every single time a client



```
(5) 0xb42bf743f35e054f8daa071063a4f68ccdaac9cb
(6) 0x1fd919b07efeaec86e0175960ade291bd64585f7
(7) 0x51077d8e61c50277438f2ae506af4b65c6e3c921
(8) 0x626dce3fa898047978465921055cda14830a54d5
(9) 0x099a9587c56f0ad564774e2cbd22029ff59e4496

Private Keys
==================
(0) 6002dfa42df021cc05f3a000f40ced173782335cef6faeb46112b0ae81846803
(1) a99048a8586f92813ff6cdeb45e921b2412fe64411149b1423f08e819da80510
(2) fc77843c560dfaf7da97ec92fd2e83c13a3167149ed8e157bb3eb7a14cf55f45
(3) 61a0c30b3f4862aa5be726030798adbfe2c674196a61b6346686180a8e5efd44
(4) 0cf13691bfcebdbf3ed94745aa58bec390b336dc191370c8f4b61d7cead817a4
(5) e0d2c6e90f8b818881ed1391893b6d6f8c9110014bea93e59289c21a72ccb763
(6) 9919ad8df9b511c9fe4cdb120bcbe1969846cdda6affaa4cf824cd54e2b387c0
(7) 4e7649b7a9504a97ecbb5cf28cd6af1dbab274955d993d0e476351c86792cc96
(8) f9d51c05ab2dcd133541b23005534dc30a6c03b6fc81a809b7fcf7c38d54cbc9
(9) 0f1eff8037def58499f5646c5c5af1e28c23a0e2806f8b622ea6b4db105db695

HD Wallet
==================
Mnemonic:      tape because symptom kangaroo mango business echo ability unlock cancel wag
on empower
Base HD Path:  m/44'/60'/0'/0/{account_index}

Listening on localhost:8545
eth_compileSolidity
eth_estimateGas
eth_accounts
eth_accounts
eth_sendTransaction

  Transaction: 0xa55ff13a43322f47bd2054eda3cc8918daa563bde22a8ffa55e82365c92fe944
  Contract created: 0x16aeaa48bdaac24944ae4781686bd5f0ef450b7c
  Gas usage: 0x09bf81
  Block Number: 0x01
  Block Time: Wed May 17 2017 14:41:47 GMT+0300 (EEST)

eth_newBlockFilter
eth_getFilterChanges
eth_getTransactionReceipt
eth_getCode
eth_uninstallFilter
eth_call
```

Figure 14. Deployed contract



```
  Transaction: 0xa55ff13a43322f47bd2054eda3cc8918daa563bde22a8ffa55e82365c92fe944
  Contract created: 0x16aeaa48bdaac24944ae4781686bd5f0ef450b7c
  Gas usage: 0x09bf81
  Block Number: 0x01
  Block Time: Wed May 17 2017 14:41:47 GMT+0300 (EEST)

eth_newBlockFilter
eth_getFilterChanges
eth_getTransactionReceipt
eth_getCode
eth_uninstallFilter
eth_call
eth_accounts
eth_sendTransaction

  Transaction: 0xa7085f0ce1a1c6a5100df218fec2277a2fca5128cfde16b8d2d2b017aadd1c39
  Gas usage: 0x697b
  Block Number: 0x02
  Block Time: Wed May 17 2017 14:42:16 GMT+0300 (EEST)

eth_call
eth_accounts
eth_sendTransaction

  Transaction: 0x134f5637229fe5deb9d7f9becff15b4ceff91aef52c01c732637c35ff8be838d
  Gas usage: 0x69bd
  Block Number: 0x03
  Block Time: Wed May 17 2017 14:42:23 GMT+0300 (EEST)

eth_call
eth_accounts
eth_sendTransaction

  Transaction: 0x3e25e95a6a098b352b4ca73720cbafd8ce27b854060138b367722f5c7eac9775
  Gas usage: 0xb848
  Block Number: 0x04
  Block Time: Wed May 17 2017 14:42:39 GMT+0300 (EEST)

eth_call
eth_accounts
eth_sendTransaction

  Transaction: 0x641b4278c40ede27ea520eb6b2a66540c011481990a530bfa6ddf2d0d25846f9
  Gas usage: 0xb841
  Block Number: 0x05
```

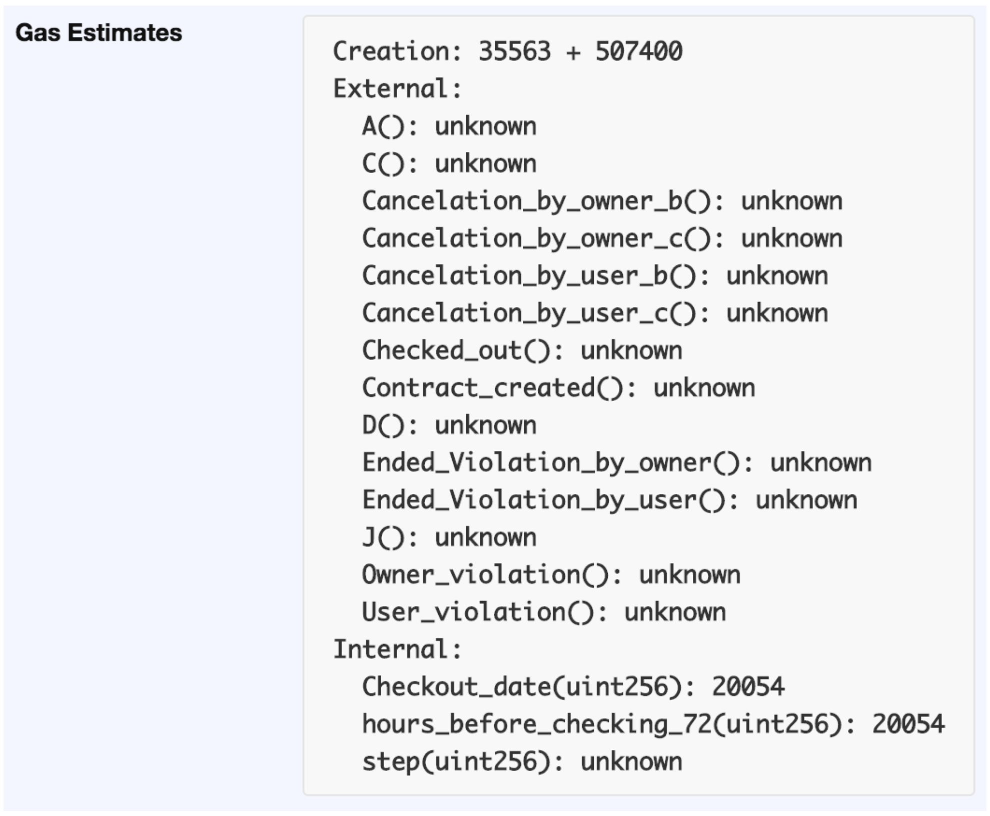
Figure 15. Transactions(Call a function)

wanted to rent a place. Because the deployment of a contract cost much higher than just running it(calling different functions). As Figure 16 can show the creation and the transaction resulting into create of the smart contract is much higher than calling one of the functions .i.e. `checkout_date()`.

The table 1 and Figure 16 shows how significant is the difference between the smart contract deployment and a function execution.

	Gas estimation
Contract deployment and execution	$35,563 + 507,400 = 542,963$
Function execution	$20,054$
Difference in %	3.69 %

Table 1. Gas estimation


Gas Estimates

```
Creation: 35563 + 507400
External:
  A(): unknown
  C(): unknown
  Cancelation_by_owner_b(): unknown
  Cancelation_by_owner_c(): unknown
  Cancelation_by_user_b(): unknown
  Cancelation_by_user_c(): unknown
  Checked_out(): unknown
  Contract_created(): unknown
  D(): unknown
  Ended_Violation_by_owner(): unknown
  Ended_Violation_by_user(): unknown
  J(): unknown
  Owner_violation(): unknown
  User_violation(): unknown
Internal:
  Checkout_date(uint256): 20054
  hours_before_checking_72(uint256): 20054
  step(uint256): unknown
```

Figure 16. Deployment and Execution Gas estimation

Due to this major difference in gas consumption the wise choice is to deploy a contract once but run multiple instances of the same contract running in parallel. Therefore, at the deployment time of the smart contract we create array of bit vectors which is empty at first then it get populated each time a instance of the smart contract is instantiated alongside a unique *ID* [Web+16] assigned to each contract. Each bitvector is just a token, in other words it's just an `uint` witch cost almost nothing. In conclusion the host can have multiple guest all using the same smart contract and saving a lot of ether by just running one business process smart contract.

4.3 Discussion

Although some services *AirBnB* and its likeness exist, running a collaborative business process like `chainurent` on the blockchain can be proven highly less expensive, much more profitable and highly secured. Since the core of the whole application is running on etheruem it is therefore a peer-to-peer for one and it runs on a public ledger secondly which makes it difficult to cheat and almost impossible to hack. It eliminates the needs of having to involves banks and other services like *Mastercard*[6], *Visa*[7] or third party payment services in the likeness of *Alipay*[8], *PayPal*[9] or *Stripe*[10]. Running on ethereum also guarantees no downtime since ethereum is a supercomputer virtually running on millions of computers.

In addition to that the timer service allows to enforce business rules by enabling or disabling some branches of business process before or after a specified time. These are backed up but the blockchain architecture, which as said previously is temper-proof. This very important in terms of collaboration because it allows full transparency and shifts the trust from people-to-people to the blockchain.

[6]https://www.mastercard.us/en-us.html
[7]http://www.visa.com/globalgateway/gg_selectcountry.jsp
[8]https://intl.alipay.com/
[9]https://www.paypal.com/
[10]https://stripe.com/

5 Conclusions and Future work

In this work, we describe the extension of the prototype of a BPMN execution engine that runs on top of Ethereum. More precisely, we extended the execution engine with the support to timer events found between tasks in sequence and also timer events as part of a deferred choice pattern. The approach consisted in adding time-guards inside the functions that emulate the execution of tasks and events that were affected by the occurrence of a timer event in the model. In this way, the guards would replace the need for an explicit external timer service. Thus, the solution rely on the analysis of the topology of the input model and the use of such information in the generation of the Solidity code. We implemented the approach and integrated the model analysis and code generation within the infrastructure of a BPMN execution engine. Finally, we tested the prototype with a few sample BPMN models.

5.1 Future Work

Because the timer event can also be a start event, a boundary event we could also cover those cases. e.g. When a timer is a boundary event, it could be interrupting or non-interrupting. In the case that it is interrupting we would have to remove the token and move it to the next node. Moreover, if the timer event is non-interrupting, the opposite will occur, on token will be placed on the node following the timer event without ever removing the token at the position at which the timer event is being a boundary. Moreover, if the timer event is a start event we have to find a way to set time in case the process a occur once or repeatedly.

On the other hand, we can also add a javascript service to the engine so that it take care of the time externally whenever a timer event is triggered. In that case, the external javascript service would keep counting the passing time and not just take a timestamp then compares it with another one later, but instead count the time and execute a given node at once.

Some work on the engine is also needed, in some cases when a process has a higher amount of nodes(tasks) the compilation time increase exponentially. This compilation slowdown occurs when the process executes in testing mode. So, if a computer low in memory it could crush the computer; a feasible solution is to separate the backend and the frontend and push all of the heavy liftings at the backend.

References

[Biz] Bizagi. *BPMN 2.0 by example*. URL: `%7Bhttp://resources.bizagi.com/docs/BPMNByExampleENG.pdf%7D`.

[But13] Vitalik Buterin. *Ethereum White Paper*. 2013. URL: `https://github.com/ethereum/wiki/wiki/White-Paper`.

[Cama] Camunda. *BPMN 2.0 Implementation Reference*. URL: `https://docs.camunda.org/manual/7.3/api-references/bpmn20/`.

[Camb] Camunda. *Camunda docs*. URL: `%7Bhttps://docs.camunda.org/manual/7.6/reference/bpmn20/events/timer-events/%7D`.

[DB17] Institute for Development and Research in Banking Technology. "Optimized Execution of Business Processes on Blockchain". In: *Application of Blockchain techonlogy to bankning and Financial Sector in India* (2017). URL: `http://www.idrbt.ac.in/assets/publications/Best%5C%20Practices/BCT.pdf`.

[DR13] Marlon Dumas and Marcello La Rosa. *Foundamentals of Business Process Management*. Springer, 2013.

[Etha] Ethereum. *Ethereum alarm clock*. URL: `http://www.ethereum-alarm-clock.com/`.

[Ethb] Ethereum. *Serpent programming language*. URL: `%7Bhttps://github.com/ethereum/wiki/wiki/Serpent%7D`.

[Ethc] Ethereum. *Solidity prgraming language*. URL: `%7Bhttp://solidity.readthedocs.io/en/latest/%7D`.

[F09] Pfitzner K.and Decker G.and Kopp O.anf Leymann F. *Web Service Choreography Configurations for BPMN*. Springer, 2009. ISBN: 978-3-540-93851-4.

[Kos+14] Felix Kossak et al. *A Rigorous Semantics for BPMN 2.0 Process Diagrams*. Springer, 2014. ISBN: 978-3-319-09930-9.

[LWR14] Andreas Lanz, Barbara Weber, and Manfred Reichert. “Time patterns for process-aware information systems”. In: *Springer* (2014).

[Mer] Piper Merriam. *Ethereum Alarm Clock Documentation*. URL: https://media.readthedocs.org/pdf/ethereum-alarm-clock-service/latest/ethereum-alarm-clock-service.pdf.

[Nak09] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2009. URL: https://bitcoin.org/bitcoin.pdf.

[OMG10] OMG. *BPMN 2.0 by Example*. 2010. URL: http://www.omg.org/spec/BPMN/20100601/10-06-02.pdf.

[OMG11] OMG. *Business Process Model and Notation(BPMN)*. 2011. URL: http://www.omg.org/spec/BPMN/2.0/PDF/.

[Ora] Oracle. *Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*. URL: https://docs.oracle.com/cd/E29542_01/doc.1111/e15176/intro_bpm_studio_bpmpd.htm#BPMPD143.

[RLP16] Marcello La Rosa, Peter Loos, and Oscar Pastor. *Business Process Management*. Springer, 2016.

[SH99] August-Wilhelm Scheer and Michael Hoffmann. “From Business Process Model to Application System - Developing an Information System with the House of Business Engineering (HOBE)”. In: *Springer* (1999).

[Web+16] Ingo Weber et al. “Untrusted Business Process Monitoring and Execution Using Blockchain”. In: *Proc. of BPM*. Vol. 9850. LNCS. Springer, 2016, pp. 329–347.

[WG] Peter Y. H. Wong and Jeremy Gibbons. *A Relative Timed Semantics for BPMN*. URL: http://www.cs.ox.ac.uk/peter.wong/pub/bpmntime.pdf.

Appendix

I. Glossary

```
pragma solidity ^0.4.0;

contract Process_0g2814d_Contract {
    uint tokens = 1;
    uint subprocesses = 0;
    uint hours_before_checking_72_time = 0;
    uint Checkout_date_time = 0;

    function hours_before_checking_72(uint localTokens) internal
        returns (uint) {
        hours_before_checking_72_time = now;
        return localTokens & uint(~2) | 4;
    }
    function Owner_violation() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 296 != 296)
            return false;
        if (now >= 4 seconds + Checkout_date_time )
            return false;
        step( localTokens & uint(~296) | 64 );
        return true;
    }
    function Ended_Violation_by_owner() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 64 != 64)
            return false;
                step( localTokens & uint(~64) | 0 );
        return true;
    }
    function Checked_out() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 128 != 128)
            return false;
                step( localTokens & uint(~128) | 0 );
        return true;
    }
    function Ended_Violation_by_user() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 512 != 512)
            return false;
                step( localTokens & uint(~512) | 0 );
        return true;
    }
    function Checkout_date(uint localTokens) internal returns (uint)
       {
        Checkout_date_time = now;
        return localTokens & uint(~16) | 256;
    }
    function User_violation() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 296 != 296)
            return false;
        if (now >= 4 seconds + Checkout_date_time )
            return false;
        step( localTokens & uint(~296) | 512 );
        return true;
    }
    function Cancelation_by_owner_c() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 2048 != 2048)
            return false;
                step( localTokens & uint(~2048) | 0 );
        return true;
    }
    function Cancelation_by_user_c() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 8192 != 8192)
            return false;
                step( localTokens & uint(~8192) | 0 );
        return true;
    }
    function Cancelation_by_owner_b() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 5124 != 5124)
            return false;
        if (now >= 3 seconds + hours_before_checking_72_time )
            return false;
        step( localTokens & uint(~5124) | 2048 );
        return true;
    }
    function Cancelation_by_user_b() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 5124 != 5124)
            return false;
        if (now >= 3 seconds + hours_before_checking_72_time )
            return false;
        step( localTokens & uint(~5124) | 8192 );
        return true;
    }
    function A() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 32768 != 32768)
            return false;
                step( localTokens & uint(~32768) | 65536 );
        return true;
    }
    function Contract_created() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 1 != 1)
            return false;
                step( localTokens & uint(~1) | 32768 );
        return true;
    }
    function C() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 65536 != 65536)
            return false;
                step( localTokens & uint(~65536) | 16384 );
        return true;
    }
    function J() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 296 != 296)
            return false;
        if (now < 4 seconds + Checkout_date_time )
            return false;
        step( localTokens & uint(~296) | 128 );
        return true;
    }
    function D() returns (bool) {
        uint localTokens = tokens;
        if (localTokens & 5124 != 5124)
            return false;
        if (now < 3 seconds + hours_before_checking_72_time )
            return false;
        step( localTokens & uint(~5124) | 131072 );
        return true;
    }

    function step(uint localTokens) internal {
        bool done = false;
        while (!done) {
            if (localTokens & 2 != 0) {
                localTokens = hours_before_checking_72(localTokens);
                continue;
            }
            if (localTokens & 131072 != 0) {
                localTokens = localTokens & uint(~131072) | 56 ;
                continue;
            }
            if (localTokens & 16 != 0) {
                localTokens = Checkout_date(localTokens);
                continue;
            }
            if (localTokens & 16384 != 0) {
                localTokens = localTokens & uint(~16384) | 5122 ;
                continue;
            }
            done = true;
        }
        tokens = localTokens;
    }

    function getActivatedFlowNodes() returns (uint) {
        uint flowNodes = 0;
        uint localTokens = tokens;
        if (localTokens & 2 == 2)
            flowNodes |= 1 ;
        if (localTokens & 8 == 8)
            flowNodes |= 4 ;
        if (localTokens & 64 == 64)
            flowNodes |= 8 ;
        if (localTokens & 128 == 128)
            flowNodes |= 16 ;
        if (localTokens & 512 == 512)
            flowNodes |= 32 ;
        if (localTokens & 16 == 16)
            flowNodes |= 64 ;
        if (localTokens & 32 == 32)
            flowNodes |= 128 ;
        if (localTokens & 2048 == 2048)
            flowNodes |= 256 ;
        if (localTokens & 8192 == 8192)
            flowNodes |= 512 ;
        if (localTokens & 1024 == 1024)
            flowNodes |= 1024 ;
        if (localTokens & 4096 == 4096)
            flowNodes |= 2048 ;
        if (localTokens & 32768 == 32768)
            flowNodes |= 8192 ;
        if (localTokens & 1 == 1)
            flowNodes |= 16384 ;
        if (localTokens & 65536 == 65536)
            flowNodes |= 32768 ;
        if (localTokens & 256 == 256)
            flowNodes |= 65536 ;
        if (localTokens & 4 == 4)
            flowNodes |= 131072 ;
        return flowNodes;
    }
```

```
}
```

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public