Process Digitalization using Blockchain: EU Parliament Elections Case Study

Marek Skotnica[1][a], Marta Aparício[2][b], Robert Pergl[1][c] and Sérgio Guerreiro[2][d]
[1]*Faculty of Information Technology, Czech Technical University, Prague, Czech Republic*
[2]*Department of Computer Science, Instituto Superior Técnico, Lisbon, Portugal*

Keywords: Blockchain, Smart Contracts, Digitalization, Elections.

Abstract: Blockchain aims to be a disruptive technology in many aspects of our lives. It attempts to disrupt the aspect of our lives that were hard to digitize, such as democratic elections or were digitized in a centralized way, such as the financial system or social media. Despite the initial success of blockchain platforms, many hacks, design errors, and a lack of standards was encountered. This paper aims to provide a methodical approach to the design of the blockchain-based systems and help to eliminate these issues. The approach is then demonstrated in an EU parliament elections case study.

1 INTRODUCTION

The Blockchain smart contracts promise a way to digitize processes where value and trust are involved. This applies to processes in many industries, such as supply chain management, finance, government process automation, voting, and many others. However, there are many challenges to overcome before essential processes can be deployed to the Blockchain. For example, after deploying a smart contract to the Blockchain, the smart contract cannot be updated, and possible errors cannot be fixed. The errors may result in significant monetary or security damage. Therefore, a methodical approach to design, validate, implement, and simulate is needed to address the problem.

This paper describes a method to digitize processes using Blockchain technology based on the Business Process Management (BPM) method and the Model-Driven Engineering (MDE) approach. It is expected that by applying this method, the programming errors will be reduced.

To demonstrate the functionality of the proposed approach, a EU parliament elections case study was created. A model of a EU parliament elections voting process running in Blockchain was modeled, a Blockchain smart contract was generated, and finally, a simulation of the functionality was performed.

The paper is organized as follows: In Section 2, the research method and the research question are formulated. A related research is covered in Section 3. In Section 4, the underlying scientific foundations are briefly discussed. An approach to digitize processes using the Blockchain technology is introduced in Section 5. The proposed approach is demonstrated on a EU parliament elections case study in Section 6. In Section 7, the current results are summarized, and further research is proposed.

2 RESEARCH QUESTION AND METODOLOGY

The research question is: **How to digitize processes using blockchain smart contracts in a methodical way and eliminate programming errors while avoiding a dependency on a particular blockchain implementation?**

The design science research approach is applied in this paper as described in (Hevner et al., 2004; Hevner, 2007), which is shown in Figure 1. In the first cycle, a relevance of the problem is discussed in Section 1. The environment in which this research is applied is model driven engineering. The challenge is the application of the MDE to the blockchain technology. The environment in which this research is

[a] https://orcid.org/0000-0002-8811-3389
[b] https://orcid.org/0000-0002-1857-0381
[c] https://orcid.org/0000-0003-2980-4400
[d] https://orcid.org/0000-0002-8627-3338

Skotnica, M., Aparício, M., Pergl, R. and Guerreiro, S.
Process Digitalization using Blockchain: EU Parliament Elections Case Study.
DOI: 10.5220/0010229000650075
In *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2021)*, pages 65-75
ISBN: 978-989-758-487-9

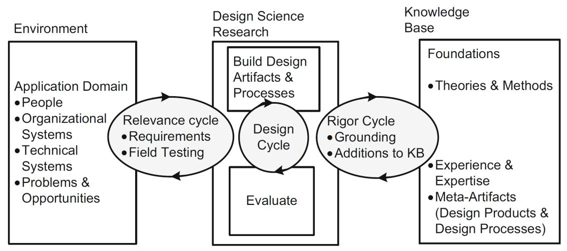


Figure 1: Design Science Research Cycles (Hevner, 2007).

applied is electronic voting.

The second cycle is described in Section 6 and Section 7. A complex case from the domain is modelled and simulated as a blockchain smart contract.

And, finally, the relevant grounding into the existing knowledge base is in Section 5, Section 6. The Section 3 provides an overview of existing approaches, and the Section 5 shows how to apply the MDE approach to the blockchain technology.

3 RELATED RESEARCH

There have been attempts at raising the level of abstraction from Code-Centric to Model-Centric Smart Contracts development. The different approaches tried so far can be divided into three: the Agent-Based Approach, the State Machine Approach, and the Process-Based Approach (Aparício. et al., 2020). For this research and going further into state of the art already mentioned in (Skotnica and Pergl, 2020), directly related to this paper, the main focus will be the Process-Based Approach.

In (Weber et al., 2016) was proposed a tool to support inter-organizational processes through Blockchain technology. To ensure that the joint process is correctly executed, the control flow and business logic of inter-organizational business processes are compiled from the processes models into Smart Contracts. Weber et al. developed a technique to integrate Blockchain into the choreography of processes to maintain trust, employing triggers and web services. By storing the status of process execution across all involved participants, as well as to coordinate the collaborative business process execution in the Blockchain. The validation was made against the ability to distinguish between conforming and non-conforming traces. (García-Bañuelos et al., 2017) presented an optimization in regards to the already presented paper (Weber et al., 2016). To compile BPMN models into a Smart Contract in Solidity Language, the BPMN model is first translated into a reduced Petri Net. Only after this first step, the reduced Petri is compiled into a Solidity Smart Contract. Compared to (Weber et al., 2016), (García-Bañuelos et al., 2017) managed to decrease the amount of paid resources and achieve higher throughput. Caterpillar, first presented in (Pintado, 2017) and further discussed in (López-Pintado et al., 2018), is an open-source Business Process Management System that runs on top of the Ethereum Blockchain. Like any BPMS, Caterpillar supports the creation of instances of a process model (captured in BPMN) and allows users to track the state of process instances and to execute tasks thereof. The specificity of Caterpillar is that the state of each process instance is maintained on the Ethereum Blockchain, and the workflow routing is performed by Smart Contracts generated by a BPMN-to-Solidity compiler. Given a BPMN model (in standard XML format), it generates a Smart Contract (in Solidity), which encapsulates the workflow routing logic of the process model. Specifically, the Smart Contract contains variables to encode the state of a process instance, and scripts to update this state whenever a task completes or an event occurs. (Tran et al., 2018) presented a tool that automatically creates a well-tested Smart Contract code from specifications that are encoded in the business process and data registry models based on the implemented model transformations. The BPMN translator can automatically generate Smart Contracts in Solidity from BPMN models while the registry generator creates Solidity Smart Contract based on the registry models. The BPMN translator takes an existing BPMN business process model as input and outputs a Smart Contract. This output includes the information to call registry functions and to instantiate and execute the process model. The registry generator takes data structure information and registry type as fields, and basic and advanced operations as methods, from which it generates the registry Smart Contract. This work builds upon already seen works, such as (García-Bañuelos et al., 2017; Weber et al., 2016), for the BPMN translation algorithms.

Going into further detail over the case study domain, highly used as a case study in research (Horcas et al., 2014), voting is a fundamental part of democratic systems. The current voting system raises concerns for those who depend on it to elect their representative, particularly concerning integrity; security; transparency; and accessibility, which translates into low turnout. E-voting by itself doesn't address all these concerns due to requiring supervision by a central authority. However, combining the E-voting solution with the Blockchain technology could be a solution, since the latter is decentralized, and distributed (Curran, 2018).

In (Dagher et al., 2018) was proposed a

blockchain-based voting system, named BroncoVote, that preserves voter privacy and increases accessibility, while keeping the voting system transparent, secure, and cost-effective. BroncoVote used the smart contracts in Ethereum blockchain to keep a record of every user in the system as well as all the ballots and the information regarding them. The smart contracts were also used to achieve access control. Besides, integrating Paillier homomorphic encryption into the system to preserve voter privacy. By resorting to Ethereum's blockchain and smart contracts, they enabled voters administration and auditable voting records. However, it is important to keep in mind that, in Ethereum any computation requested has to be paid according to a preset deterministic unit called gas, which has a variable real-world cost. Thus reducing the computation performed on the blockchain is imperative to reduce the cost of a service running off the blockchain (Azzopardi et al., 2020). As in (Dagher et al., 2018), (Khoury et al., 2018) also suggests the usage of blockchain technology to solve trust issues. It suggests a new business model for voting service providers, where the voting service provider enables the voting event organizers to deploy an event voting smart contract. The event management server deploys in the Ethereum network the voting contracts configured according to the voting event customer. (Zhang et al., 2019) presents an Ethereum based electronic voting protocol, Ques-Chain, that may surpass the voting domain to other fields with similar needs. This protocol responds to problems of three entities, for the e-voting systems tackles the threat of malicious manipulation by hackers, for the service providers helps to prevent and eliminate scams, given the high costs to perform data cleaning, at last, for the voters tackles the doubts that may arise about the integrity of the voting procedure and anonymity failure. The protocol presented can be divided roughly into four stages: Setup Stage, where the organizer initialize the e-voting; Sign Stage, where the voter will get a signed-blinded-ballot from the organizer; Vote Stage, where the voters vote; Count Stage, where the voters will count legal ballots and publish the results. There are some similarities between the last work mentioned and (Al-Rawy and Elçi, 2018). However, the latter divides the voting process into six phases. Both resort to blockchain technology and public-private key systems. At last, in (Bellini et al., 2018), a process-centric blockchain-based architecture, called HyperVote, is discussed. The advantage of HyperVote over the other papers here presented is that the first follows the business trend of cloud computing, virtually turning it into an online service (XaaS). Meaning in a more practical manner that HyperVote is able to support dynamic selection, deployment, and execution of an e-voting process whose requirements are tailored to the user needs. It is a pay-per-use configurable approach bringing on costs during the limited timeframe of an election. This allows to optimize the allocation of resources as the same infrastructure can serve different customers in different time frames. The HyperVote is an XML based data structure containing all the information required by the execution environment to deploy, execute, account, and monitor the service. The HyperVote binds a Workflow Model with the concrete Atomic Services (chaincode) and Hyperledger Network structure and Cloud Infrastructures that will be invoked and selected by the workflow, respectively, when executed.

4 METHODOLOGIES USED

4.1 BPM

A process is a collection of events, activities, and decisions that collectively lead to an outcome that brings value to an organization's customers. Understanding and managing these processes to ensure that they consistently produce value is critical for the effectiveness and competitiveness of an organization. Business Process Management is a body of principles, methods, and tools to discover, analyze, redesign, implement, and monitor business processes. Process models and performance measures are foundational pillars for managing processes. Since these pillars help to achieve business goals efficiently and effectively while complying with boundary requirements for governance, risk, and compliance.

There are many languages for modeling business processes diagrammatically, perhaps one of the oldest are flowcharts. Nowadays, there is a widely-used standard for process modeling, namely the Business Process Model and Notation (BPMN). BPMN is an industry-standard for workflow procedures, supported by the Object Management Group (OMG).

4.2 MDE

One of the promising methods that can facilitate the development of Smart Contracts is Model-Driven Engineering. Through Model-Driven Engineering, executable code can be generated for Smart Contracts from a set of models that specify the processes to be supported. For instance, (Horcas et al., 2014) and (Mavridou and Laszka, 2018) are tool-supported methods that allow the generation of Solidity (Ethereum Foundation, 2020) Smart Contracts from BPMN diagrams and finite state machines, respectively.

Using Model-Driven Engineering requires a set of models to be used as input for code generation. In particular, rather than focusing on algorithms and the elaboration of code, it focuses on the creation of models as representations of the software to be built. These models are then used to be transformed into other models or code. Key aspects of this transformational approach are that it speeds up the creation of software, that it enhances software quality, and that it facilitates the portability and interoperability of software. In *"Towards a shared ledger business collaboration language based on data-aware processes"* (Hull et al., 2016) called for the development of a shared ledger business collaboration language, which should be a language that business people can understand and use in cross-organizational business processes supported by blockchain technology.

4.3 Blockchain

(Nakamoto, 2009) described the Blockchain as an architecture that enables participants to perform electronic transactions without relying on trust. This makes it possible that each block contains some data, the hash of that block, and the hash of the previous block. The data stored in a block corresponds to multiple transactions, and each transaction has an identification for the sender, receiver, and asset. The block also has a hash, which identifies its content and is always unique. If something is changed within the block, it will cause the hash value to change. This is why hashes are useful for detecting changes in blocks. The hash value of the previous block effectively creates a Blockchain, and it is this technology that makes the Blockchain so secure. However, the hashing technology is far from enough, as the high computing power that exists today, among which the computer can calculate thousands of hashes per second. To mitigate this problem, the Blockchain has a consensus mechanism called Proof-of-Work. This mechanism slows down the creation of new blocks since if a block is tempered, the Proof-of-Work of all the previous blocks has to be recalculated. Therefore, the security of Blockchain comes from its creative use of hashing and a Proof-of-Work mechanism. Of course, its distributed nature also adds a certain degree of security because the Blockchain does not use a central entity to manage the chain, but uses a peer-to-peer network that anyone can join.

As a new technology, Blockchain raises questions regarding its viability. As (Garther, 2019) claims, Blockchain technology is still very immature to support most of the potential use cases and is still a tremendous amount of research, implementation, and adoption to be done. Further, building systems on Blockchain is non-trivial due to the steep learning curve of the Blockchain technology. According to a survey by (Gartner, 2018), *"23 percent of [relevant surveyed] CIOs said that blockchain requires the newest skills to implement any technology area, while 18 percent said that blockchain skills are the most difficult to find."*.

4.4 Smart Contracts

Blockchain is run by machines that essentially perform three things: networking, consensus, and state management. These machines must run application-layer software responsible for updating the state. For Blockchains like Ethereum (Vitalik Buterin, 2013), a Turing-complete virtual machine is the application layer. The software programs that developers deploy to the Blockchain virtual machine are usually called Smart Contracts. The Ethereum Blockchain was specifically created and designed to support Smart Contracts. Solidity is a high-level programming language to implement Smart Contracts specially design for the Ethereum Virtual Machine (EVM). the smart contracts are deployed onto the blockchain in the form of a specialized bytecode. This bytecode then runs on each Ethereum node in EVM. Since creating smart contracts directly in the bytecode would be too impractical, multiple specialized high-level languages have been created, together with the compilers needed to convert them into EVM bytecode.

The smart contract itself is written in any simple language, depending on which Blockchain will be deployed, and is usually very short. Although they are relatively short, they are extremely powerful as they leverage the huge power of cryptography and blockchains since they operate across organizations and between individuals. Smart contracts can store records of who owns what, as well as promises without the need for intermediaries or exposing people to fraud. These contracts can automatically execute instructions given in the past. For pure digital assets, there is no setback, because the value to be transferred can be locked in the contract when the contract is created, and automatically released when the conditions and terms are met. In this case, fraud is impossible because the program can truly control the assets involved without the need for a trusted intermediary. Like all algorithms, Smart Contracts may require input values and only act if certain predefined conditions are met. When a particular value is reached, the Smart Contract changes its state and executes the functions, that are programmatically predefined algorithms, automatically triggering an event on the Blockchain. If false data is entered into the system, then false results will be outputted.

4.5 Tokens

From what has been written throughout this paper, the use of Blockchain technology seems the future to manage digital assets, due to its security and immutability features. In (Voshmgir, 2019) tokens are described as ways to represent *”programmable assets or access rights, managed by a smart contract and an underlying distributed ledger. They are accessible only by the person who has the private key for that address and can only be signed using this private key.”*

However, without the use of fungible tokens, this would be impossible to do since no unique information can be written into the token. To represent tokens that have to be unique and hold information instead of values, non-fungible tokens are preferred. On one hand, the fungible tokens can be exchanged to any other token of the same type. Non-fungible tokens, on the other hand, cannot be replaced by other non-fungible tokens of the same type. For these reasons, the fungible tokens are uniform, meaning they are identical to one another, where the non-fungible tokens are unique. Fungible tokens are divisible into smaller units, while the non-fungible tokens cannot be divided. In this paper, the non-fungible tokens are used to represent voter’s ballots.

Due to the need to standardize the general structure of Ethereum Blockchain tokens, Ethereum Improvement Proposals (Ethereum, 2020) was created. These standards allowed for Smart Contracts running on Ethereum to interact with tokens defined by other Smart Contracts. One of these standards is the ERC-20, providing a typical list of rules on how the fungible tokens should be structured. That makes them easy to trade, as there is no need to differentiate the tokens. To represent unique assets, however, the tokens must be non-fungible, even if they are of the same type. In order to facilitate these types of tokens, the ERC-721 standard was created (Voshmgir, 2019).

4.6 Das Contract

The Das Contract (Skotnica and Pergl, 2020) is a DSL designed to be an implementation-independent language for specifying blockchain smart contracts. The Das Contract specification is based on a subset of BPMN 2.0 (OMG, 2011), UML class diagram (OMG, 2015) and extended with concepts specific to the blockchain execution environment. The

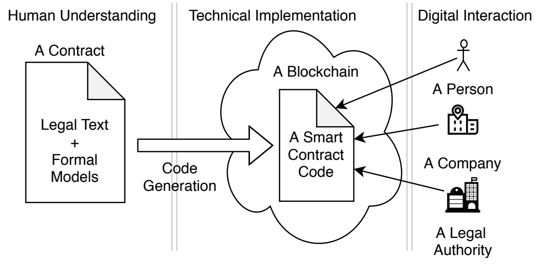


Figure 2: A proposed concept architecture of DasContract (Skotnica and Pergl, 2020).

extension allows to specify off-chain forms to interact with the smart contract, smart contract logic, and ability to work with payments or tokens. The language is still under development, and it will be enhanced with the support of decentralized identity and data oracles.

According to the (Skotnica and Pergl, 2020) and shown in Figure 2, DasContract’s approch consists of three parts:

Human Understanding: part defines a contract between multiple parties that they need to agree on. Such a contract is a combination of legal text and formal ontological models. The legal text in some form specifies the legal validity of the formal model.

Technical Implementation: part specifies how formal models from the contract are transformed into a software executable code and uploaded into a blockchain as a smart contract.

Digital Interaction: is a part where people, companies and legal authorities can interact with the agreed upon contracts. Since the contract is in a blockchain, the interaction is fully digital, and thanks to cryptography can also be legally binding. Blockchain by design also provides an audit trail of all actions performed by the parties and ensures that the agreed upon contract is executed correctly.

5 OUR APPROACH

Our approach was designed to answer the research question from section 2. It adopts the BPM methodology to provide a methodical way of designing a blockchain smart contract. A Model-driven engineering approach is used to generate a smart contract source code for a particular blockchain. In this paper an Ethereum smart contract platform is used, but the approach can be generalized. An application of our approach is shown in section 6 on a process of EU parliament elections.

The BPM life-cycle in context of digitizing a process to be used in blockchain technology is following:

1. Design - A process is designed in a DasContract language with blockchain limitations in mind. The major limitation compared to a traditional software system is it's immutability and a requirement that all performed actions are deterministic.
2. Modeling - A simulation of the DasContract may be performed to ensure the correct behaviour of the process. This may be critical because the contract can't be changed after being deployed and it may be handling significant monetary or legal value.
3. Execution - A smart contract source code is generated and uploaded to the blockchain. Because the metamodel is implementation independent, any supported blockchain platform can be chosen.
4. Monitoring - Due to the inherent blockchain capability to record all transaction history, auditing and analysis of process execution history can be done.
5. Optimization - The optimization is not relevant in blockchain because once the smart contract is uploaded it is not possible to change it.
6. Re-engineering - Similar to the optimization, the re-engineering is not available. A new process can be designed and uploaded to the blockchain but the old one will be still running.

5.1 Das Contract to Smart Contract Transformation

The DasContract consists of three interconnected models: process model, data model, and forms model. The metamodel of the data and forms models is shown on Figure 3. The highlighted parts in blue are the recent additions to support the representation of fungible and non-fungible tokens and enums. The tokens inherit from the entity, and therefore they can have data properties to represent token metadata.

Process Model: specifies the contract's process activities, their execution order, property mappings, and user roles. The model is based on an extended subset of the BPMN 2.0 level 3 notation. The main addition is support for blockchain tokens (described in Section 4.5) that allows to issue and receive both fungible and non-fungible tokens. During the transformation, the process sequence is transformed into a smart contract programming language statements such as if-else, functions, etc. This process may vary for different blockchain implementations.

Data Model: is a domain model of the process is based on the UML class diagram. It allows specifying entities, properties, and relationships between them. The properties may contain primitive types such as int, bool, and string, but arrays and enumerations are supported as well. Address and Address-Payable types are added to support the storage of a blockchain actor's addresses and consequent token or cryptocurrency transfers. A blockchain token is represented as a special type of entity and supports defining both fungible and non-fungible tokens with custom properties. A transformation of the models to the smart contract source code is straightforward because the blockchain programming languages support such concepts in the form of classes, structures, enums, and properties. The support of tokens is native on some platforms. On other platforms such as Ethereum is added as a third-party library.

Forms Model: specifies an interface for the user activity input. The forms model is generated into two parts during the generation – off-chain model and on-chain code that is similar to a web application client-side and server-side code. The off-chain model is interpreted by a blockchain wallet that allows the user to interact with the smart contract. Most of the blockchain implementations currently have no support for off-chain code inside a smart contract. The on-chain code handles validations, user rights, and property bindings.

6 CASE STUDY: EU ELECTION

Since 1979, occurred to date, nine Elections of the European Parliament, once every five years according to the universal adult suffrage. From 2020, 704 Members of the European Parliament will be elected by more than 400 million eligible voters from 27 member states, for this reason it is considered the second largest democratic elections in the world (Hare, 2015). Each member state has its own voting system, either being by Preferential voting, Closed lists or Single Transferable Vote, all can be combined with Multiple Constituents. Further, each member state has its own voting methods for citizens resident abroad and if whether or not voting is compulsory. Each member state has different rules determining who can vote for and run as a Member of the European Parliament. Although this is not a very standardized process, the European Parliament is the only institution in the European Union, directly elected by the citizens. So, it is extremely important that this

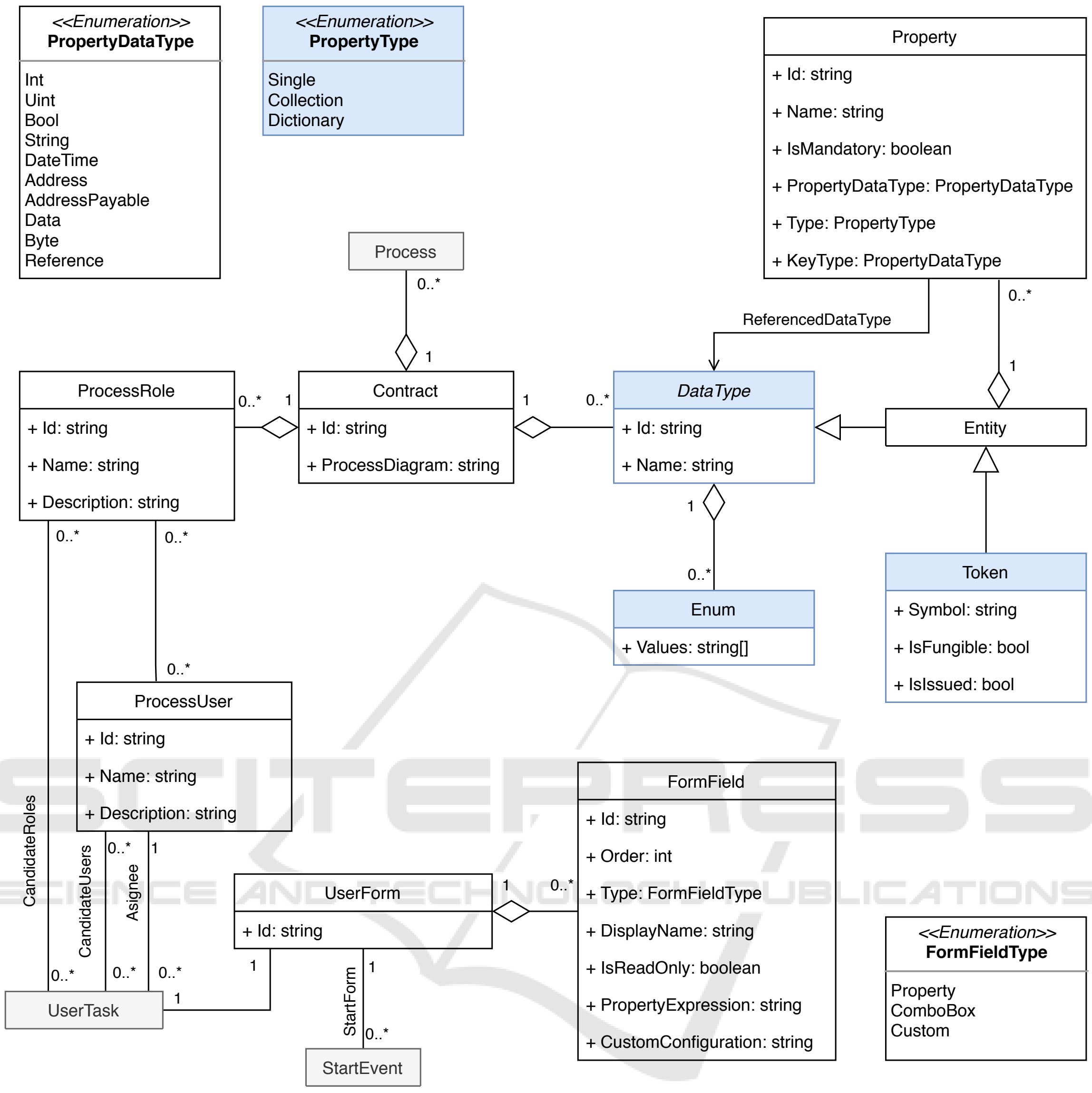


Figure 3: A DasContract Data and Forms Metamodel.

is a transparent, meddle-proof, usable, authenticated, accurate and verifiable process (Patil et al., 2013).

The development and implementation of constitutional and legal provisions have been one of the engineering concerns. Since for instance, the Elections of the European Parliament have the estimated cost of €700 million. Blockchain has been the most promising technology when it comes to the electoral domain, seeing that per each vote, a new transaction, if valid, is added to the end of the blockchain and remains there forever (Curran, 2018). For this solution, no centralized authority is needed to approve the votes, and everyone agrees on the final tally as they can count the votes themselves, as anyone can verify that no votes were tampered with and no illegitimate votes were inserted.

This case study shows how to digitize the EU election process by following a method proposed in Section 5. A simplified version of the process was designed and tested on the Ethereum blockchain.

6.1 Process Design

The EU election process is very complex. The EU issues general guidelines on how country elections should look like, and each country then implements its legislation to describe how the elections are done in a particular country. This means that each of the 27 EU countries does this process differently. To avoid this complexity, it was assumed that there is a unified

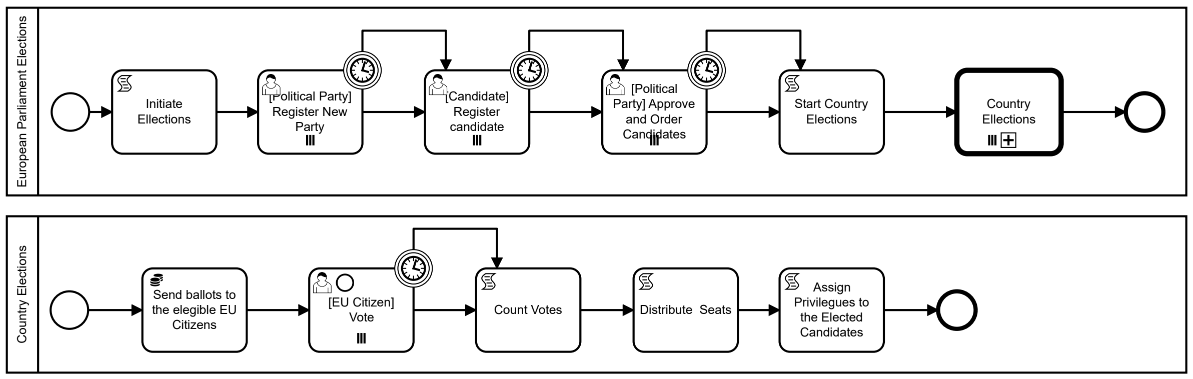

European Parliament Elections
Initiate Elections
[Political Party] Register New Party
[Candidate] Register candidate
[Political Party] Approve and Order Candidates
Start Country Elections
Country Elections
Country Elections
Send ballots to the elegible EU Citizens
[EU Citizen] Vote
Count Votes
Distribute Seats
Assign Privilegues to the Elected Candidates

Figure 4: A DasContract Process Model of the EU Elections.

voting process and each country votes according to one of the three voting systems – preferential voting, closed lists, and single transferable vote.

The modeled process is shown on Figure 4. It starts with an initiation of the elections where all the countries and their voting systems are initiated in the contract. After the initiation, political parties are able to register until a set deadline is expired. Later, candidates are able to register, and in the next step, the political parties approve the candidates. In some countries, candidates without countries are allowed. After a deadline for the approval is reached, the country elections for each of the initiated countries are started in parallel as a subprocess.

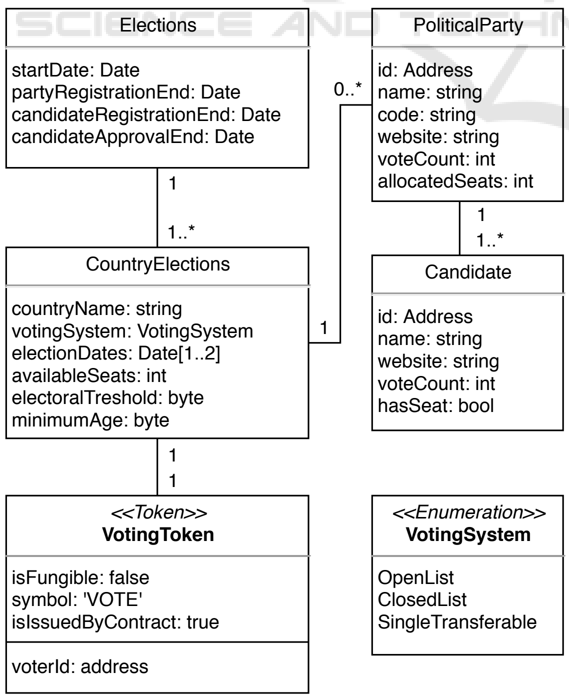


Figure 5: A DasContract Data Model of the EU Elections.

The country elections subprocess starts by creating sending non-fungible tokens to eligible EU voters. The non-fungible tokens represent voting ballots to assure that a double vote is prevented. In the next step, the EU citizens are able to vote by sending their tokens to the election smart contract during the election days. After the voting is over, votes are counted, and seats are distributed according to the country's voting system. In the end, privileges are assigned to the selected candidates.

The process model also contains a validation and script task that is currently entered in form of a solidity programming language. In the future, a domain-specific language is expected to be designed and used.

The data model is shown on Figure 4. It only contains the essential information about the elections because the storage costs are high on public blockchain because of a need for replication on all nodes and keeping all the history. A new concept added in this paper is a possibility to specify tokens and enumerations in the data model. The voting token representing a ballot is specified as a non-fungible token and to assure it's uniqueness, it is associated with an individual voter identifier. The voter identifier represents a citizen's identity public key, however, the identification of the citizen is outside of the scope of this paper and is a subject of further research.

The forms model rendering for the process step *Vote* is shown on Figure 6. The DasContract currently generates only an on-chain validation code because the off-chain code is not supported on Ethereum platform.

6.2 Execution

During the execution step, the designed model was used to generate a Solidity smart contract according to the MDE principles. The generation algorithm is available on GitHub under an open-source license (Skotnica, 2020).

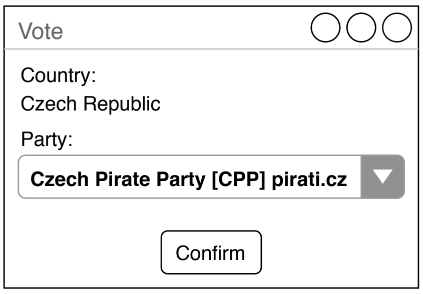
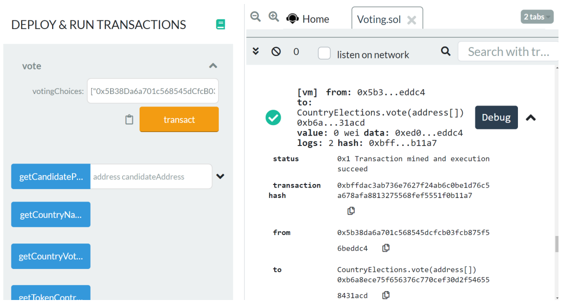


Figure 6: A User Interface of the Closed List Vote in a Blockchain Wallet.



```
[vm] from: 0x5b3...eddc4
to:
CountryElections.vote(address[])
0xb6a...31acd
value: 0 wei data: 0xed0...eddc4
logs: 2 hash: 0xbff...b11a7
status        0x1 Transaction mined and execution
              succeed
transaction   0xbffdac3ab736e7627f24ab6c0be1d76c5
hash          a678afa8813275568fef5551f0b11a7
from          0x5b38da6a701c568545dcfcb03fcb875f5
              6beddc4
to            CountryElections.vote(address[])
              0xb6a8ece75f656376c770cef30d2f54655
              8431acd
```

Figure 7: A Screenshot from Remix Simulation.

An interesting example of the generated code shows an implementation of the vote function that accepts a non-fungible voting token and a voting choices cast by the citizen:

```
function vote(address[] memory votingChoices)
  public {
require(votingTokensContract
  .isEligibleToVote(msg.sender),
    "Voter not eligible to vote");
require(isVotingOpen(),
  "Voting is currently not allowed");
require(votingChoices.length > 0,
  "At least one cadidate must be chosen");

if(votingSystem == VotingSystem.ClosedList){
  require(
    politicalPartiesMap[votingChoices[0]]
      .exists,
    "Party address is invalid");
  politicalPartiesMap[votingChoices[0]]
    .voteCount++;
}
else if(votingSystem ==
  VotingSystem.OpenList){
    ...
}
else if(votingSystem ==
  VotingSystem.SingleTransferable){
    ...
}

votingTokensContract
  .transferVoteToken(msg.sender);
}
//The generated code was adjusted to improve
//it's readability.
```

The name of the function is taken from the task name in a process model. The parameters of the vote functions are from the forms model that is associated with a user task. The address of a voting citizen and his voting ballot are not the parameters to prevent people from acting as someone else. The citizen's id is taken from the *msg* object that contains a verified public key by the original sender. Inside the function, the first step is to validate the person's eligibility to vote in the current election (E.g. he can be a minor and have no voting rights.). The second validation assures that the correct process step is selected. Finally, the votes are validated so the citizen does not lose his ballot without voting for a valid candidate.

After the validations, a vote is counted according to the country voting system and the citizen's vote is transferred back to the smart contract to prevent a double vote. Note that the casting vote is counted but not forgotten due to blockchains' inherent history keeping. The blockchain also guarantees that the function is processed one at a time so the *voteCount++* is safe to be used even when it is called from multiple sources in parallel.

In the end, the generated source code was simulated in the Ethereum Remix simulation environment. A screenshot from the simulation showing the vote function is show in Figure 7.

6.3 State of the Art

Currently, the election process can be designed, simulated, and executed in blockchain technology. However, the public blockchain implementation Ethereum capacity only allows to process up to fifteen transactions per seconds (Blockchair, 2020) for all of it's users and an average transaction price as of September 16th 2020 is 4.301 USD (YChart, 2020). The two reasons alone would make it impossible to run the real EU parliament elections on Ethereum blockchain. It is expected that in the future the transaction prices will go lower and the transactions per second will increase due to implementation of the proof of stake consensus algorithm and second layer scaling solutions. These scaling limitations do not apply to private blockchain solutions such as Hyperledger Fabric (The Hyperledger White Paper Working Group, 2018) and therefore it would be possible to execute the EU election process there. However, the private blockchains may not guarantee the same security properties as the public blockchains because the network is run by a handful of selected operators.

6.4 Summary

Following the proposed method allowed us to model the EU elections case in a visual editor using a standardized BPMN language. The DasContract subset of the BPMN language only allowed for concepts that are valid in Blockchain. Furthermore, the data model was expressed, and a custom non-fungible token was designed. It was possible to test the process model's behavior during the design using standardized BPMN simulation tools.

During the Execution phase, a Solidity smart contract was generated using an open-source algorithm. The generated code contains 365 lines, and it was executed in the Remix environment. Due to Ethereum transaction fees, it would not make economic sense to run it on millions of users.

Overall, the proposed approach seems to make the digitalization of processes in the blockchain environment easier as it is based on a standardized process modeling notation. The generation of smart contract code reduces programming errors. However, only Ethereum smart contracts can be generated, but the DasContract model seems to be transferable to other blockchain implementations.

7 CONCLUSIONS AND FURTHER RESEARCH

This paper introduced a method to digitize processes using blockchain smart contracts based on the MDE approach. The method followed the BPM steps and was demonstrated in the EU parliament elections case study. It was shown that using the DasContract visual domain-specific language it is possible to design, model, execute and monitor the process. By following the method, a possibility to introduce a programming error in code was reduced because the code was generated. Finally, the generated code was simulated on the Ethereum blockchain.

The limitation of this approach was also mentioned, mainly that there is currently no public blockchain capable of supporting millions of transactions in a cost and time-efficient way. According to the Gartner (Gartner, 2019), production-ready blockchain technology will be available by 2030. By then, the transaction fees should be lower, so significant cost savings can be realized.

7.1 Further Research

- Extend the example and DasContract language with a Decentralized Identity (DiD) (W3C, 2020) and Blockchain oracles such as ChainLink (Steve Ellis, et a., 2017).
- Compare possibilities of executing the processes in public vs private blockchains.
- Case studies and usability studies to improve the proposed method.
- Compare different blockchain implementations such as Cardano (IOHK, 2017), Tron (Tron Foundation, 2018), EOS (Daniel Larimer, et al., 2017), and Hyperledger Fabric (The Hyperledger White Paper Working Group, 2018) and their ability to execute DasContract models.
- Transformation of the models can be formalized using the transformation specification languages like QVT (OMG, 2016) and ATL (OMG, 2002).

ACKNOWLEDGEMENTS

This research has been supported by CTU SGS grant No. SGS20/209/OHK3/3T/18.

REFERENCES

Al-Rawy, M. and Elçi, A. (2018). A design for blockchain-based digital voting system.

Aparício., M., Guerreiro., S., and Sousa., P. (2020). Towards an automated demo action model implementation using blockchain smart contracts. In *Proceedings of the 22nd International Conference on Enterprise Information Systems - Volume 2: ICEIS,*, pages 762–769. INSTICC, SciTePress.

Azzopardi, S., Colombo, C., and Pace, G. J. (2020). A technique for automata-based verification with residual reasoning. In *MODELSWARD*, pages 237–248.

Bellini, E., Ceravolo, P., Bellini, A., and Damiani, E. (2018). Designing process-centric blockchain-based architectures: A case study in e-voting as a service. In *Data-Driven Process Discovery and Analysis*, pages 1–23. Springer.

Blockchair (2020). Ethereum transactions per second.

Curran, K. (2018). E-voting on the blockchain. *The Journal of the British Blockchain Association*, 1(2):4451.

Dagher, G. G., Marella, P. B., Milojkovic, M., and Mohler, J. (2018). Broncovote: Secure voting system using ethereum's blockchain.

Daniel Larimer, et al. (2017). EOS.IO Technical White Paper v2.

Ethereum (2020). Ethereum Improvement Proposals (EIPs).

Ethereum Foundation (2020). The Solidity Contract-Oriented Programming Language.

García-Bañuelos, L., Ponomarev, A., Dumas, M., and Weber, I. (2017). Optimized execution of business processes on blockchain.

Garther (2019). The Reality of Blockchain.

Gartner (2018). Gartner survey reveals the scarcity of current blockchain deployments. https://www.gartner.com/en/newsroom/press-releases/2018-05-03-gartner-survey-reveals-the-scarcity-of-current-blockchain. Accessed: 2020-08-10.

Gartner (2019). The reality of blockchain.

Hare, P. W. (2015). *Making diplomacy work: intelligent innovation for the modern world*. CQ Press.

Hevner, A. (2007). A Three Cycle View of Design Science Research. *Scandinavian Journal of Information Systems*, 19(2).

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. *MIS Q.*, 28(1):75–105.

Horcas, J., Pinto, M., and Fuentes, L. (2014). An aspect-oriented model transformation to weave security using cvl. In *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 138–150.

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

IOHK (2017). Cardano Blockchain.

Khoury, D., Kfoury, E., Kassem, A., and Harb, H. (2018). Decentralized Voting Platform Based on Ethereum Blockchain.

López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., and Ponomarev, A. (2018). CATERPILLAR: A Business Process Execution Engine on the Ethereum Blockchain. *CoRR*, abs/1808.03517. _eprint: 1808.03517.

Mavridou, A. and Laszka, A. (2018). Designing secure ethereum smart contracts: A finite state machine based approach. In *International Conference on Financial Cryptography and Data Security*, pages 523–540. Springer.

Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*.

OMG (2002). ATLAS - Authorization Token Layer Acquisition Service, version 1.0.

OMG (2011). Business Process Model and Notation (BPMN), Version 2.0.

OMG (2015). Unified Modeling Language, version 2.5.

OMG (2016). QVT - MOF Query/View/Transformation, version 1.3.

Patil, M., Pimplodkar, V., Zade, A. R., Vibhute, V., and Ghadge, R. (2013). A survey on voting system techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(1):114–117.

Pintado, O. (2017). Caterpillar: A Blockchain-Based Business Process Management System.

Skotnica, M., e. a. (2020). Dascontract 2.0 github repository.

Skotnica, M. and Pergl, R. (2020). Das Contract - A Visual Domain Specific Language for Modeling Blockchain Smart Contracts. In Aveiro, D., Guizzardi, G., and Borbinha, J., editors, *Advances in Enterprise Engineering XIII*, pages 149–166, Cham. Springer International Publishing.

Steve Ellis, et a. (2017). ChainLink - A Decentralized Oracle Network.

The Hyperledger White Paper Working Group (2018). An Introduction to Hyperledger.

Tran, A. B., Lu, Q., and Weber, I. (2018). Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management. In *BPM*.

Tron Foundation (2018). Tron - Advanced Decentralized Blockchain Platform.

Vitalik Buterin (2013). Ethereum Whitepaper.

Voshmgir, S. (2019). *Token economy how blockchains and smart contracts revolutionize the economy*. Blockchain HUB, Berlin.

W3C (2020). Decentralized Identifiers (DIDs) v1.0.

Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., and Mendling, J. (2016). Untrusted business process monitoring and execution using blockchain. In La Rosa, M., Loos, P., and Pastor, O., editors, *Business Process Management*, pages 329–347, Cham. Springer International Publishing.

YChart (2020). Ethereum average transaction fee.

Zhang, Q., Xu, B., Jing, H., and Zheng, Z. (2019). Queschain: an ethereum based e-voting system. *arXiv preprint arXiv:1905.05041*.