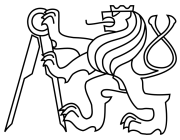
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



# ASSIGNMENT OF MASTER'S THESIS

| | |
|---|---|
| **Title:** | Using Blockchain Smart Contracts in the DEMO Methodology |
| **Student:** | Bc. Barbora Hornáčková |
| **Supervisor:** | Ing. Marek Skotnica |
| **Study Programme:** | Informatics |
| **Study Branch:** | Web and Software Engineering |
| **Department:** | Department of Software Engineering |
| **Validity:** | Until the end of summer semester 2017/18 |

## Instructions

Blockchain 2.0 (BC) introduces smart contracts that help build digital decentralized autonomous organizations. Enterprise engineering (EE) discipline may help this technology with formal ontological foundations. The aim of the thesis is to assess whether the BC together with the EE can provide a foundation for more secure and democratic enterprises; especially in the banking industry.
Steps to follow:
* Review BC to evaluate what this technology is capable of.
* Evaluate compatibility of BC and EE.
* Propose a software architecture of an IT system based on BC and EE.
* Compare the proposed SW architecture with the state-of-the-art of business process management systems based on BPMN.
* Create a prototype of an exemplary banking process in Ethereum Solidity as a basis for further discussion.
* Evaluate possible benefits of application EE principles in BC implementation.

## References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague December 29, 2016

Czech Technical University in Prague
Faculty of Information Technology
Department of software engineering science

Master's thesis

# Using Blockchain Smart Contracts in the DEMO Methodology

***Bc. Barbora Hornáčková***

Supervisor: Ing. Marek Skotnica

3rd January 2018

# Acknowledgements

I would like to express my gratitude to everyone who supported me and helped me to complete this thesis. Mostly, I would like to thank to my supervisor Marek Skotnica for his time and guidance, but also to my family and friends, namely Ondrej Husár and Eliška Veverková.

# Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 3rd January 2018 .....

Czech Technical University in Prague
Faculty of Information Technology


### Citation of this thesis

Hornáčková, Barbora. *Using Blockchain Smart Contracts in the DEMO Methodology.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2018.

# Abstrakt

Blockchain (BC) je nová technológia prinášajúca decentralizovanú, replikovanú, autonómnu a bezpečnú databázu. Jej najznámejšie využitie sa spája s kryptomenou bitcoin. Možné sú však aj ďalšie aplikácie, ako napríklad inteligentné kontrakty. Inteligentný kontrakt je transakcia uložená v blockchaine, ktorá obsahuje spustiteľný kód a má vlastné interné úložisko. Uloženie kontraktu do blockchainu zaručuje jeho nezmeniteľnú exekúciu a uchovávanie dát. Inteligentné kontrakty umožňujú vytváranie komplexnejších a decentralizovaných aplikácií (Dapps), či decentralizovaných autonómnych organizácií (DAOs). Enterprise Engineering (EE) je vedecká disciplína, ktorá skúma všetky aspekty organizácií od podnikových procesov, informačných a technických zdrojov, až po organizačné štruktúry. EE zahrňuje DEMO metodológiu, ktorá sa zaoberá modelovaním a návrhom organizácií. Z toho dôvodu sú blockchain a inteligentné kontrakty predmetom záujmu EE a ich možného prepojenia pre podporu tvorby Daaps.

Cieľom tejto práce je preskúmať EE, možnosti blockchainu, a zároveň inteligentných kontraktov. Následne je vyhodnotená kompatibilita DEMO metodológie a inteligentných kontraktov, na základe čoho sú predstavené dva prístupy k použitiu týchto kontraktov v danej metodológií. Následne je vytvorená metóda pre tvorbu inteligentných kontraktov na základe DEMO modelov a návrh architektúry informačného systému založeného na EE s použitím blockchainu. Pre overenie realizovatelnosti vytvorenej metódy je vypracovaná ukážková implementácia inteligentného kontraktu s použitím DEMO metodológie v procese získania a splácania hypotéky.

**Klíčová slova** Modelovanie podnikových procesov, DEMO, DEMO metodológia, blockchain, blockchain 2.0, inteligentné kontrakty

# Abstract

Blockchain (BC) is a new technology that introduces a decentralized, replicated, autonomous and secure databases. It is mostly known for its use with bitcoin, but it has more applications beyond that, such as smart contracts. Smart contract (SC) is a transaction embedded to blockchain that contains executable code and its own internal storage, offering immutable execution and record keeping. Smart contracts enable the creation of more complex decentralized applications (Dapps) and even decentralized autonomous organizations (DAOs) on blockchain. Enterprise Engineering (EE) examines all aspects of organizations from business processes, informational and technical resources, to organizational structure. Its underlying DEMO methodology focuses on modeling and designing organizations. Therefore, blockchain and smart contracts have been subject of interest concerning the discipline of Enterprise Engineering (EE) and how they can be used together, enhancing the creation of Dapps.

Thus, this thesis reviews the discipline of EE and the capabilities of blockchain and smart contracts. Based on that, the compatibility of DEMO methodology and blockchain smart contracts are evaluated and two approaches of using smart contracts in the DEMO methodology are introduced. Consequently, a method of creating smart contracts from DEMO models is formed and a software architecture of an IT system based on EE integrating smart contracts is proposed. Finally, a proof of concept implementation of a smart contract of a mortgage process using the DEMO methodology is developed, to demonstrate the feasibility of proposed concepts.

**Keywords** Enterprise Engineering, DEMO, DEMO methodology, blockchain, blochchain 2.0, smart contract

# Contents

# List of Figures

# List of Tables

# Introduction

Business processes are crucial parts of any company or enterprise, mainly because every product is an outcome of a collection of actions. Understanding the processes and their relations is an important step for the organization of any business. Information technologies (IT) play a significant role in business processes or processes in general, as more and more of them are supported by IT systems. Not only are business processes essential to understand the operation of a business on an organizational level, but they are also inevitable for designing a flexible IT system [6]. Therefore, business process management (BPM) and business process management systems (BPMS) have received a lot of attention over the last years. Business process management should help to connect the gap between the organizational aspects of business and the information technologies applied [6].

DEMO (Design and Engineering Methodology for Organizations) is one of the processes modeling methodologies. It is based on the scientific discipline of Enterprise Engineering, which focuses on designing enterprises. DEMO methodology brings theoretical foundation and methodology for modeling organizations. The DEMO models fulfill the C4 quality criteria (Concise, Coherent, Consistent and Comprehensive) [3] and the methodology is used for Model Driven Engineering (MDE)[7] approach for creating BPMS directly from the DEMO models, such as DEMO engine [8].

As the progress of information technologies is very dynamic, their application in IT systems should constantly be evaluated. One of the new technologies, which is a subject of many discussions concerning IT systems and EE, is a blockchain. Blockchain is mostly known as the underlying technology of bitcoin, but since its introduction, there has been a wide variety of applications. Blockchain is a decentralized, replicated and secure database running on a peer-to-peer network. Due to the solutions it brings to problems such as the double-spend and Byzantine Generals‘ Problem, blockchain has been called a breakthrough in the computer science [9]. Blockchain 2.0 enhances application of blockchain beyond cryptocurrencies and introduces concepts for flexible and

programmable transactions referred to as smart contracts. Smart contracts enable the creation of more complex decentralized applications (Dapps) and even decentralized autonomous organizations (DAOs) on blockchain.

Therefore, blockchain and smart contracts have been subject of interest concerning the discipline of Enterprise Engineering (EE) and the usage of smart contracts in the DEMO methodology, enhancing the creation of Dapps.

### Goal and Research Questions

The main goal of this thesis is to form a method of using blockchain smart contracts in the DEMO methodology. DEMO methodology presents a system to capture the essence and model processes. Blockchain, on the other hand, presents a new technology which can be used for implementation of processes. The underlying intention is to determine the possible cooperation between them in the IT system. Based on the goal of the thesis following research questions have been defined:

1. What are the blockchain technology and smart contracts capable of?
2. Is the DEMO methodology compatible with the blockchain technology?
3. How can blockchain be used in the implementation of an enterprise information system (EIS) based on DEMO methodology?
4. How to create a smart contract based on DEMO models?
5. What would be the software architecture of an EIS integrating smart contracts?

### The structure of the thesis

The thesis is organized as follows:

- **Chapter 1** provides theoretical and technical foundations about Enterprise Engineering, DEMO modeling method, Blockchain, Blockchain 2.0, smart contracts and Ethereum, needed to gain the understanding of the topic.
- **Chapter 2** covers common smart contract misconceptions, to prevent mistakes in our theory, evaluates the compatibility of blockchain and DEMO methodology and introduces two approaches for using smart contracts.
- **Chapter 3**describes a method of creating smart contracts from DEMO models, introduces a software architecture of an IT system based on Enterprise Engineering and Blockchain and describes the interaction between an IT system and blockchain, compares the proposed software

architecture to the state-of-the art business process management based on BPMN and its smart contract integration.

- **Chapter 4** introduces a proof of concept implementation of smart contract based on DEMO models of a mortgage process in Ethereum Solidity and simulation of the contract execution.
- **Conclusion** summarizes the results, evaluates the benefits of the proposed solution and mentions further research.

Chapter 1

# Theoretical Foundations

## 1.1 DEMO Methodology

DEMO means "Design and Engineering Methodology for Organizations". Its an enterprise modeling methodology for designing organizations developed by Jan Dietz and others. Demo is based on the Organization Essence Revealing (OER) paradigm and the $\psi$-Theory (PSI, Performance in Social Interaction) about organizations.

### 1.1.1 Enterprise Engineering

The term enterprise can be identified as organization, company or institution. In more general sense, it can be defined as any kind of collaborative activity by human beings [10]. They are complex systems, that besides human activity comprise also information, technology and their interaction. Dietz defines the nature of an enterprise as "enterprises are social systems, of which the operating principle consists of the ability of human beings to enter into and comply with commitments."[3]

Enterprise Engineering (EE) is the scientific discipline focused on designing whole or a part of an enterprise. It examines all aspect of the enterprise from business processes, informational and technical resources to organizational structure. EE is built on four pillars: Enterprise Ontology, Enterprise Architecture, Enterprise Governance, which all together form Enterprise Design [1].

### 1.1.2 Enterprise Ontology

Enterprise Ontology is an approach to enterprise that enhance the understanding and overview of the essence of the organization. It is the reveling of "the essence of the construction and operation of enterprise independent of its realization and implementation"[3]. The ontological model of the essence of

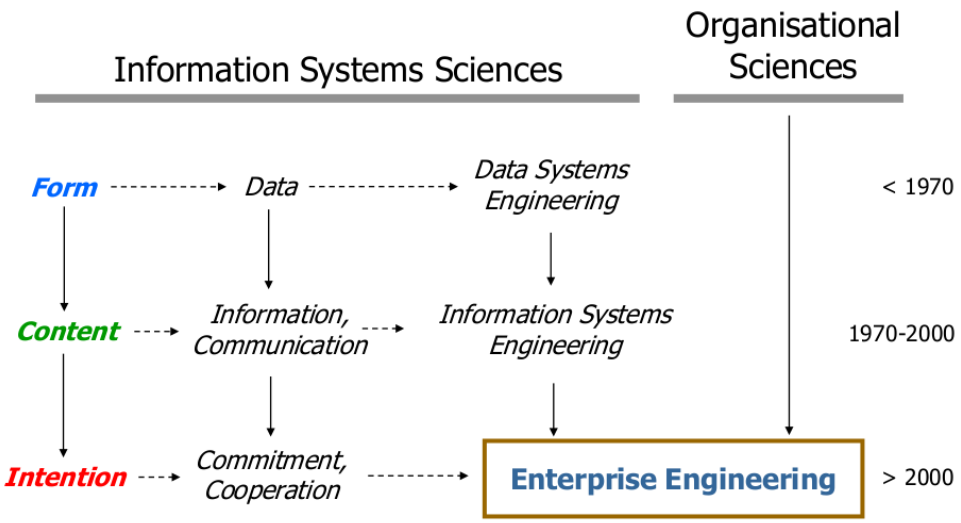



Figure 1.1: History of Enterprise Engineering [1]

organization satisfies a $C_4E$ quality criteria: Coherence, Comprehensiveness, Consistency, Conciseness, and Essence [3].

### 1.1.3 The PSI Theory

The $\psi$-Theory is the underlying theory of enterprise ontology. The theory considers organization to be a group of subjects that interact with each other in order to create certain product. The overall purpose of $\psi$-Theory is "to extract the essence of an organization from its actual appearance". The theory is composed of of four axioms: operation, transaction, composition, distinction and one theorem: organizational theorem [2].

### 1.1.4 The Operation Axiom

The operation Axiom defines that operation of an organization is based on performing acts between subjects fulfilled by actor roles. It introduces two main kinds of acts: production acts and coordination acts. Explanation of Operation Axiom was taken from thesis by Zuzana Vejrážková: [2]

> The operation axiom states that the operation of an enterprise is constituted by the activities of actor roles, which are elementary chunks of authority and responsibility. The actor roles are fulfilled by subjects, who can perform two kinds of acts: production acts and coordination acts, and these have definite results: production facts and coordination facts.

A coordination act (C-Act) is an act by which a coordination fact (C-fact) in the coordination world (C-world) is created. By performing C-acts, subjects enter into and comply with commitments towards each other regarding the performance of production acts. C-act is an act performed by one actor called the performer, and directed to another actor, called the addressee. Acts can further be divided into intention acts and proposition acts. This concept is displayed in 1.2

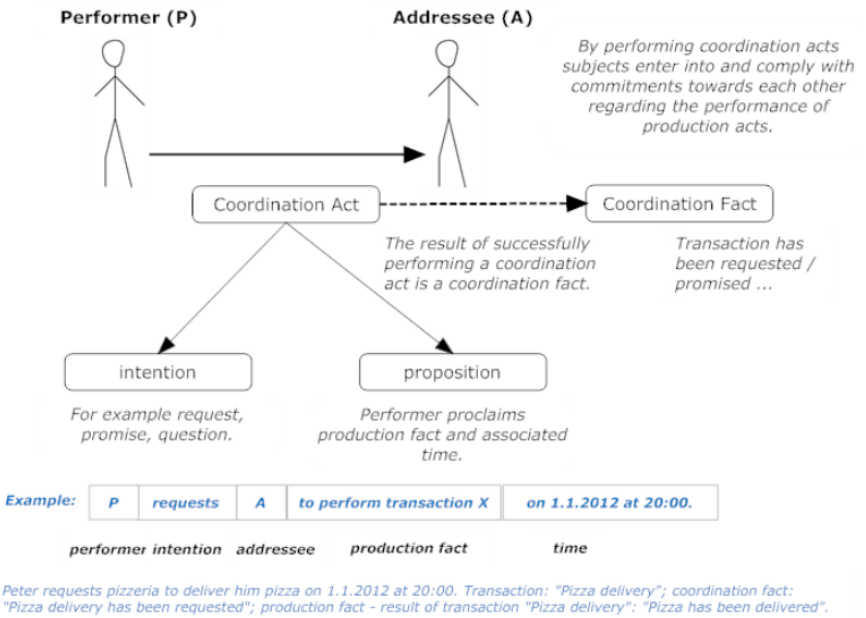



Figure 1.2: The operation axiom: relation of performer and addressee, coordination act and fact [2]

A production act (P-act) is an act by which a production fact (P-fact) in the production world (P-world) is created. By performing P-acts, the subjects contribute to bringing about goods or services that are delivered to the environment of the enterprise. C-acts are always, either directly or indirectly, about production acts and production facts, which can further be divided into material or immaterial.

Actors are the active elements of an enterprise, who operate autonomously and deal with agenda according to the existing action rules. By specifying responsibility, authority and competence, the operation axiom is related to common organizational theories. Competence is the ability of a subject to perform particular P-acts as

well as corresponding C-acts. Based on a competence, actors have authority, which they are expected to exert in a responsible way.

Based on a competence, you are given authority; if you are given authority, you have a responsibility, as displayed in figure 1.3

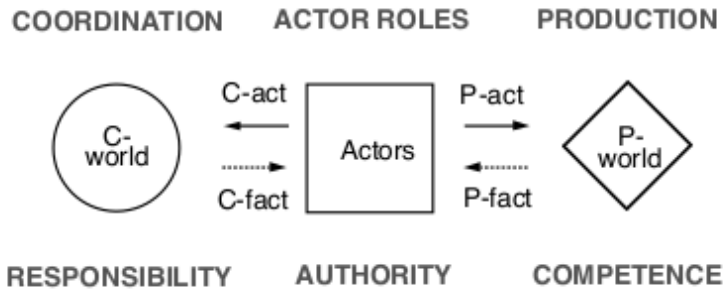



Figure 1.3: The operation axiom: relation of performer and addressee, coordination act and fact [2]

### 1.1.5 The Transaction Axiom

Transaction axiom defines the generic pattern of the occurrence of and relationship between C-Acts and P-Acts. It reveals abstraction of transaction and divides it into three phases: order, execution and result phase. Explanation of part the Transaction Axiom was taken from thesis by Zuzana Vejrážková: [2]

> The transaction axiom states that coordination acts are performed as steps in universal patterns, called transactions. Every new, original P-fact is a result of a successful transaction. Transactions always involve two actor roles (initiator and executor) and are aimed to achieve a result (P-fact). A transaction consists of three phases: the order phase (O-phase), the execution phase (E-phase) and the result phase (R-phase). The practical relevance in transaction axiom lies in detecting all C-acts, even those, that are performed tacitly. Steps are displayed in figure 1.4, notation used for transaction pattern is in Appendix C.

#### The basic transaction pattern

In the order phase, request and promise are performed. The Initiator (customer) requests a P-fact, which is created as a result of a transaction. The executor makes a promise to create this P-fact.

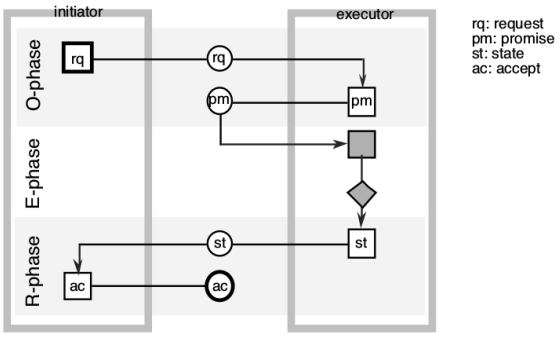



Figure 1.4: The basic pattern of a transaction [3]

In the execution phase, the execution is performed which results in creating the P-fact. In the result phase, the executor states that the fact has been created and the initiator accepts the result.

### The standard transaction pattern

Some situations might get more complicated. In some cases, the executor cannot promise creating the fact, so he declines it. Similarly, the initiator cannot accept the created fact, so he rejects it. There are three reasons for rejecting or declining, which are called validity claims (namely claim to truth, claim to justice or claim to sincerity) and are based on HabermasâĂŹ theory of communicative action.

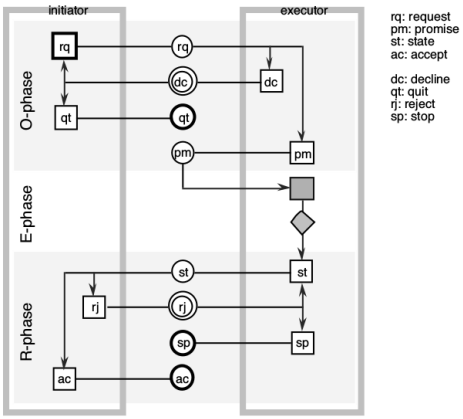



Figure 1.5: The standard pattern of a transaction [3]

### Complete transaction pattern

In real world actors might not always be successful in delivering the agreed results or might simply change their mind. The standard transaction pattern, however, does not offer such possibility resulting in deadlock situations. Therefore a revocation conversations (R-conversations) are introduced that allow actors to revert the process to some previous status by revoking one of the four basic C-acts: request, promise, state and accept. Revokes can be initiated from any status of the main transaction and repeatedly. The opposite actor of the transaction can either allow the revoke, resulting in reverting the process to the desired state or refuse the revoke keeping the current state [11].

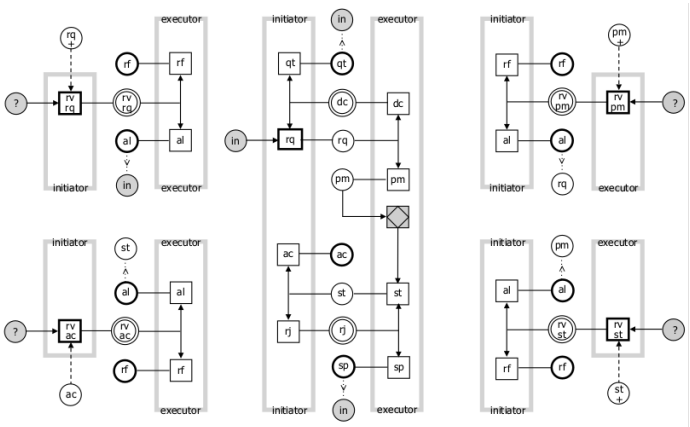



Figure 1.6: The complete transaction pattern [1]

## 1.1.6 The Composition Axiom

The Composition axiom defines the interrelations between P-facts. As Dietz states "a business process is a collection of causally related transaction types, such that the starting step is either a request performed by an actor role in the environment (external activation) or a request by an internal actor role to itself (self-activation)."[3] According to this the composition axiom states that transactions are either enclosed in some other related transactions, invoked by the customer of the organization or it can be a self-activated transaction [3].

## 1.1.7 The Distinction Axiom

The Distinction axiom defines that in the operation of actors three basic human abilities play role: performa, informa and forma. The aim of the axiom is to separate our diverse concerns and it identifies the ontological actions (performa) which is crucial for reveling the essence of the organization. Figure 1.7 further explains the three human abilities [3].

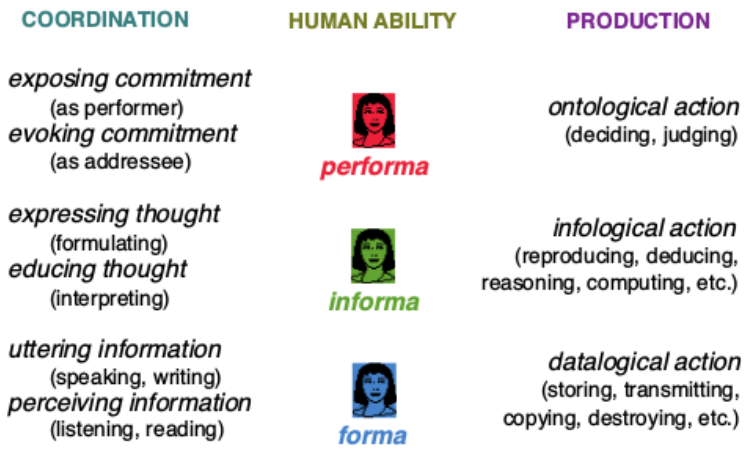



Figure 1.7: Summary of the distinction axiom [3]

### 1.1.8 The Organization Theorem

Similar to human abilities in the distinction axiom, the organizational theorem states that "the organization of an enterprise is a heterogeneous system that is constituted as the layered integration of three homogeneous systems: the B-organization (from Business), the I-organization (from Intellect), and the D-organization (from Document)."[3]. This three parts support each other from D-organization to B-organization. They are all similar in means of coordination, where actors carry out transactions in order to create production acts, but the difference is in production, where B-organization realizes the ontological production, I-organization the infological and D-organization the datalogical production [3].

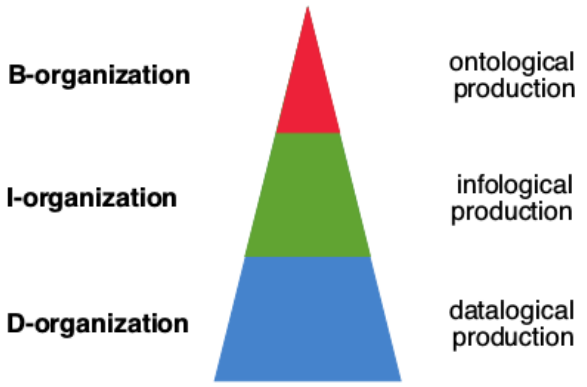



Figure 1.8: Representation of the organization theorem [3]

### 1.1.9 The Modeling Method

The DEMO modeling methodology is based on the Enterprise Ontology, therefore the modeling of organization is focused on the ontological models independent of its realization and implementation. The methodology provides four aspect models that support each other and together create the complete ontological knowledge of an organization. The aspect models are organized in triangular shape displayed in 1.9, which shows the relationship between them. Some of the aspect models consist of more than one sub models [3].

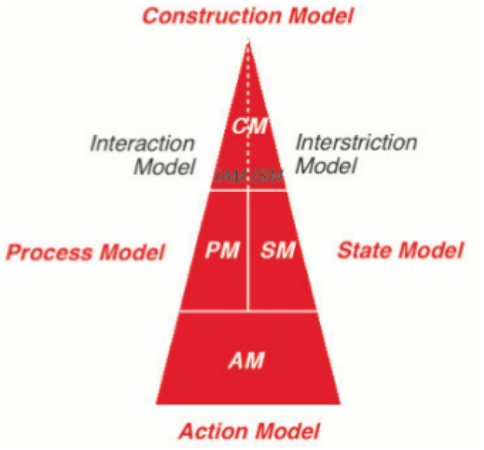



Figure 1.9: The ontological aspect models [3]

### 1.1.10 The Construction Model

The construction model (CM) is located on the top of the triangle in 1.9. It means that it is the most abstract and concise model. CM "specifies the identified transaction types and the associated actor roles, as well as the information links between the actor roles and the information banks (the collective name for production banks and coordination banks); in short, the CM specifies the construction of the organization."[3] CM is further divided into interaction model (IAM) and interstriction model (ISM). Good explanation of IAM and ISM is in thesis by Zuzana Vejrážková: [2]

#### Interaction Model

The Interaction Model shows the boundary of an organization and transaction types with identified actor roles - the initiator and the executor. It consists of one table and one diagram:

- **Transaction Product Table (TPT)** describes the identified transaction types and corresponding product types.

- **Actor Transaction Diagram (ATD)** shows relations between actor roles and transactions, includes markings of executor and initiator actor roles and organizational boundary.

#### Interstriction Model

The Interstriction model shows passive influence between actor roles. It is based on the IAM, to which the information flow is added. It contains two diagrams and one table:

- **Actor Bank Diagram (ABD)** shows relation (information links) between actor roles and information banks and actor roles and transactions. In ABD only information links are included.
- **Organization Construction Diagram (OCD)** combines the ABD and the ATD. It takes the ATD and only adds information links from the ABD to elements that are not yet connected in the ATD.
- **Bank Contents Table (BCT)** specifies the fact banks in which the elements of object classes and the instances of fact types and result types from the SM are contained.

### 1.1.11 The Process Model

The process model (PM) located in the middle of the triangle in 1.9 is more complex and detailed. It looks at the transaction patterns and also the relationships between transactions. It contains specific transaction pattern according to the $\psi$-theory for every transaction type identified in CM and it defines the causal and conditional relationships between transactions. PM shows by which transaction the process begins or ends, the order of and dependences between transactions. These two parts together specifies the state space and also the transition space of the C-world [3]. The PM model consists of: [2]

- **Process Structure Diagram (PSD)** provides structure of each process, shows process steps for every transaction and relationships between them. Steps that are not included in the PSD are not allowed. Normally, the disagreement patterns and the cancellation patterns must be included. The PSD of a business process should be understood as the complete specification of the steps in a business process that an enterprise wants to monitor or control. The PSD is the right starting point for designing the workflow support systems.

- **Information Use Table (IUT)** specifies for every object class, fact type and result type from the SM, in which steps of the PM are used its instances. It can only be created once the SM is produced.

### 1.1.12 The Fact Model

The fact model (FM) is located at the same level as PM in 1.9. The state model specifies the state of the P-world, namely, the object classes, fact and result types and ontological coexistence rules. It is located under the CM as it specifies information banks, which are used in CM [3]. The SM model consists of: [2]

- **Object Fact Diagram (OFD)** shows the relation between object classes and declaration of result types.
- **Object Property List (OPL)** describes object classes. For each object class it also describes corresponding properties and their scale. For derived fact types, derivation rules are included.

### 1.1.13 The Action Model

The action model (AM) is the lowest part of the triangle 1.9 and is the most comprehensive one. The action model defines a set of rules that serve as a guide for actors in specific actor roles in dealing with their agenda. It defines the action rules considering all the conditions and validity claims in order to carry out the transaction in different states. There are action rules for every state of the transaction (request, promise...) for every actor role in every transaction. AM is not a graphical model, it is a textual representation of action rules written in a pseudo-algorithmic language: action rule is enclosed in *on-no* bracket pair, conditions in *if-fi* and repeated actions in *do-od* pair [3].

## 1.2 Blockchain

Blockchain (BC) is a technology introduced by Satoshi Nakamoto[1] It is mostly known for its use with Bitcoin as it is its underlying technology. It is a new way of looking at transactions, assets exchange or even whole organizations. It introduces decentralized, autonomous, replicated and secure database. Based on cryptography offers trustless network with no need of intermediary, resulting in major resource and also time saving. The possibilities of applying this technology are very broad and it could be effectively used in most of the parts of our world.

### 1.2.1 Blockchain Transactions

One of the basic blockchain building block is a transaction. A blockchain transaction is a data package. Its structure changes depending on the concrete implementation of the blockchain, but it generally contains the transaction‘s data, timestamp and digital signature. In Bitcoin a transaction represents a financial transaction, exchange of possession of a bitcoin between two users or bitcoin wallets. So depending on the use case, transactions can be modeled into data structures to represent the needed type of information, weather it is an exchange of assets or contract. For example, in bitcoin “each transaction contains one or more inputs, which are debits against a bitcoin account. On the other side of the transaction, there are one or more outputs, which are credits added to a bitcoin account.”[12]

Assigning ownership of resources or assets used in blockchain transactions is done by digital signatures using public key cryptography. By using private/public key pair transactions can be signed providing a proof of ownership to whoever owns the private key. For example, in Bitcoin public key is used to receive bitcoin and private key is used to sign or ”unlock” transactions to spend those bitcoins. “When spending bitcoins, the current bitcoin owner presents his public key and a signature (different each time, but created from the same private key) in a transaction to spend those bitcoins. Through the presentation of the public key and signature, everyone in the bitcoin network can verify and accept the transaction as valid, confirming that the person transferring the bitcoins owned them at the time of the transfer.”[12]

### 1.2.2 Blocks

Each block contains number of transactions (from hundred to thousands). After certain number of transactions is gathered or last block has been created a new block is started to be created, using cryptography and Merkle trees a unique hash is calculated that represents all of the content and transactions,

[1]Satoshi Nakamoto is probably a pseudonym for either one person or a group of people, the identity is currently unknown.

furthermore each block contains hash of the previous block and a timestamp (see figure 1.10), all of these are again hashed and this hash is the representation of the block. By including the hash of the parent and timestamp, blocks are chained on "top of". This iterative process confirms the integrity of the previous block, defining the exact order and history of blocks and transactions. Thus creating the block chain's "public ledger, an ordered and timestamped record of transactions."[4] Once the transaction is contained in a block it is considered to be valid and also providing a proof-of-existence of the information carried by the transaction.

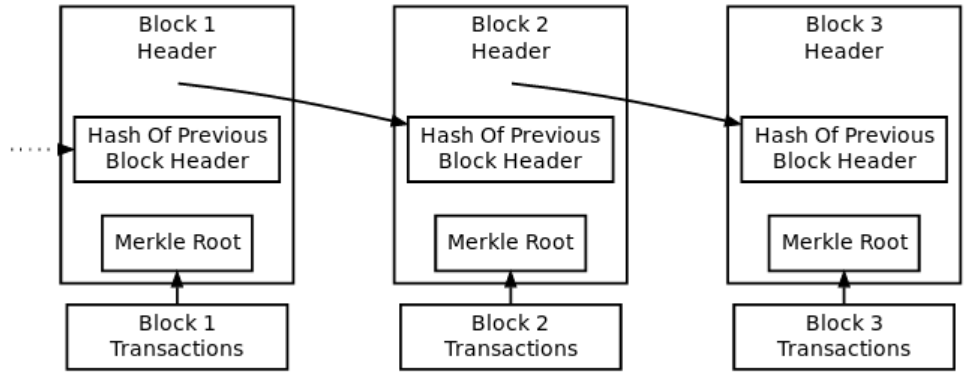



Figure 1.10: Simplified chain of blocks [4]

### 1.2.3 Network

Another building block is the blockchain decentralized peer-to-peer network, where all the nodes are equal and share the responsibility of the providing services. The nodes are connected in a mesh network with a "flat" topology [12]. There are different types of nodes in the network, some nodes have a complete copy of the blockchain or part of it, some are using APIs to obtain the information. What they all have in common, is that they are able to validate a transaction, based on some given rules and they are all connected to certain number of other nodes and exchange information between each other.

For example if some node creates a new transaction, it propagates it to its neighbors who validate the transaction and then propagate it further to their neighbors, so eventually every node receives the transaction on the best effort basis. Some nodes called miners are than able to gather all new transactions that are not in a block yet and create new blocks from them, that are again broadcasted to the network and included in the consensus blockchain.

Some nodes can have different views of the block chain, or usually just a recent history, as some of the transactions or blocks might reach them in

different order. If such a difference is encountered, nodes synchronize their chains by choosing the longer one.

### 1.2.4 Consensus algorithms

Consensus algorithm is a protocol used for creating new blocks. To provide security and trust in blockchain there are special rules under which blocks are created, so that the creation of new block is resource and time consuming, making it physically impossible to change the ledger history. This is one of the logic behind the blockchain that eliminates the need of trusted intermediary.

There are different distributed consensus protocols for creating blocks. The most known is proof-of-work. Proof-of-work, also called mining, in blockchain is based on solving a hard computational problem where the solution can be proved easily. Miners are special nodes in blockchain that are able to create new blocks and serve as timestamp servers on a peer-to-peer basis. Miners must solve the proof-of-work, for example find a value that when hashed, such as with SHA-256, the hash begins with number of zeros, the number of zeros is optimized to achieve desired difficulty as the average work required is exponential in the number of zeros. So miners try to vary the block hash by incrementing a nonce added to the block, recalculating the hash. The first miner that finds the right nonce, creates and propagates the new block and collects the reward [13].

If anyone would want to change the history of the database, lets say one block, they would need to calculate new hash of the block as it would be different with the change of its content and not only that one block but every block created after this one. This operation is considered to be so time and resource consuming, that blocks that are more than six blocks deep in the ledger are considered unchangeable. By varying the number of zeros required we can also control the frequency of creating new blocks, in bitcoin it is every ten minutes on average.

### 1.2.5 Decentralization and Distribution

Knowing the basic principles of blockchain we can review the aspects that create its key essence: decentralization, distribution and trustlessness:

- Each node can have full copy of the ledger, data quality maintained by database replication
- Every node is able to validate each transaction
- All nodes in the network are equal and connected, broadcasting transactions and blocks on the best effort basis.
- Trusted central authority replaced by computational trust in proof-of-work

- Public/Private key cryptography providing proof-of-ownership
- Timestamping serving as proof-of-existence

## 1.3 Blockchain 2.0

The architecture of blockchain was from the beginning designed to handle big variety of implementations. Cryptocurrencies, financial transactions or in general decentralization of money and payments are considerer to be the Blockchain 1.0, but further usage of blockchain is enhanced in Blockchain 2.0, aiming at general decentralization of markets and all kind of assets other than money. To put in parallel "Blockchain 1.0 has been likened to the underlying TCP/IP transport layer of the Web, with the opportunity now available to build 2.0 protocols on top of it (as HTTP, SMTP, and FTP were in the Internet model)."[9]

The main idea of Blockchain 2.0 is to use blockchains decentralized transaction ledger functionality to identify, verify and transfer all kinds of assets, contracts and properties [9]. The pool of ideas of blockchain usage is almost endless, as with its flexibility it could be used on so many different processes that we come in contact with on daily bases. Some of them are public records, properties, health care, copyrights, stocks, wills or smart contracts.

### 1.3.1 Smart Contracts

The idea of smart contracts (SC) is to offer more complex solutions than just a sell/buy transactions. Smart contract is a transaction embedded to blockchain that contains more enhanced logic, contract that is executable, has its own data storage and can access other resources to evaluate its current state and perform actions, a contract made of code. "A smart contract is a set of commitments that are defined in digital form, including the agreement on how contract participants shall fulfill these commitments."[14]

The main characteristic of a programmable smart contract are that it does not require trust between parties, as after its creation in blockchain it would be able to execute itself immutably. The parties would not need to be in further contact or use an intermediary therefore it would be autonomous. Being able to handle resources would give it self-sufficiency and given the architecture of blockchain decentralization. They are not doing something that was not possible before, just reduce the complexity of common problems and help with automation [9].

There are more possible ways of modeling contracts and processes with smart contracts, some can be performed programmatically entirely, another can use smart contract just for a part such as record keeping.

### 1.3.2 Smart Contracts Use Cases

There is a big variety of possible applications of smart contracts from finances, government to health care.

A financial use case for smart contract application could be mortgages: "Smart contracts can automate the otherwise confusing and manual process behind a mortgage contract. A smart contract in this case automatically connects the different parties involved with mortgage transactions, allowing for a frictionless and less error-prone process."[5] fig. 1.11.

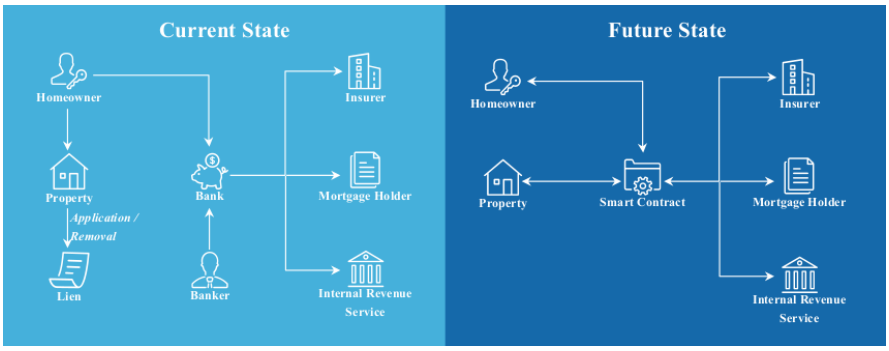



Figure 1.11: Mortgages enabled by smart contracts provide automated processing of payments and release of liens on property [5].

Government processes is one of an important suggested application of Blockchain 2.0 and smart contracts. The idea is to provide governmental services in blockchain decentralized, transparent and efficient manner. Where blockchain could for example become the database for all document records [9]. Land title recording is a nice example of using smart contract for transparent, reliable and fraudless transfer and record of land property assets fig. 1.12.

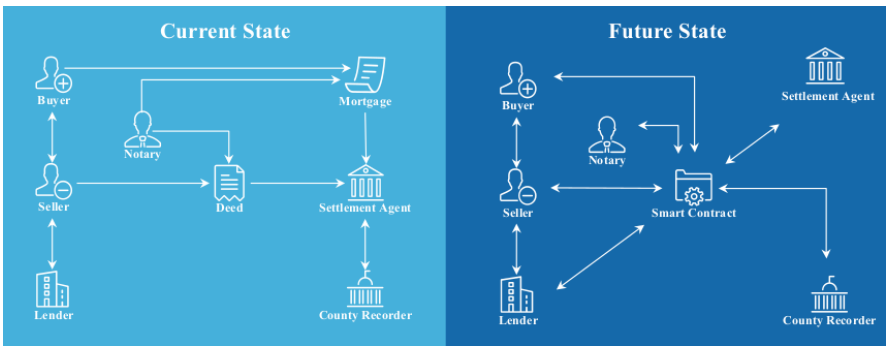



Figure 1.12: Property transfers enabled by smart contracts [5].

Another example from a health care field is a cancer research fig. 1.13.

Cancer research or in general medical research is a complicated process that involves many parties, data and consents. In this case a smart contract could serve as a common source of data to contribute to and share and transparent patient consent management process [5].

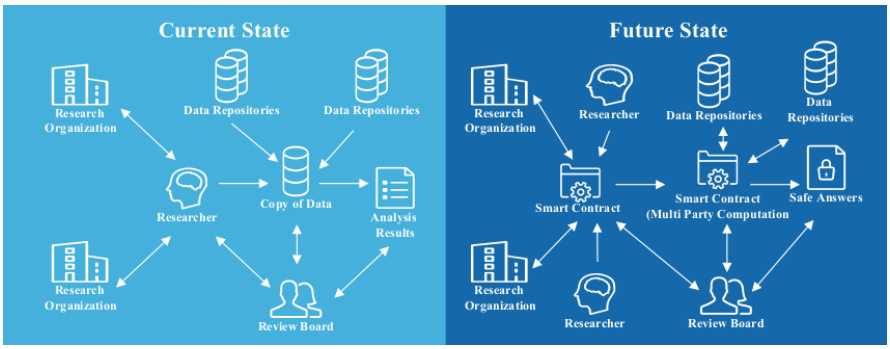



Figure 1.13: Unleashed power of data enabled by smart contracts provides more efficient data sharing across sectors and incentivizes pre-competitive collaborations [5].

### 1.3.3 Dapps, DAOs and DACs

Smart contracts have the potential not just to be simple contracts between several parties but over time they could become very complex systems involving many parties and resources. The definition of decentralized applications (Dapps) can vary but in general it refers to open source autonomous applications that use decentralized network and executes across decentralized network nodes.

When further enhancing Dapps and creating applications that handle complicated functionality, interconnect between each other and all in autonomous decentralized manner, we could create decentralized autonomous organizations (DAOs) and even decentralized autonomous corporations (DACs). DAOs and DACs are "a concept derived from artificial intelligence. Here, a decentralized network of autonomous agents perform tasks, which can be conceived in the model of a corporation running without any human involvement under the control of a set of business rules. In a DAO/DAC, there are smart contracts as agents running on blockchains that execute ranges of pre-specified or pre-approved tasks based on events and changing conditions."[9]

### 1.3.4 Private and Public Blockchain

The original intention of blockchain and bitcoin-like implementation was to create a public network, but due to some limitations it brings, private blockchains have been developed as well. The main disadvantage of public blockchain is the amount of computational power it needs in order to maintain the

ledger when used at a large scale. The second issue is the openness of the system and consequent lack of privacy of transactions and its content. The difference between public and private blockchain is based on controlling who can be part of the network, in more detail, it means who can participate in the network and in which parts, who can execute the consensus protocol and manage the ledger. It is also refereed to as permissioned blockchain, in contrast to the public blockchain, which is permissionless. It requires an invitation to join a private blockchain, where the access control mechanism may vary. [15] This means that in private blockchain there is control over the extent to which it is decentralized and anonymous [16]. Private blockchains, as there is a reduced number of processing nodes, are faster and the transaction costs might be lower[16]. On the other hand, this access control brings extra costs and complexity to the process of maintaining or joining the blockchain. There are also hybrid solutions combining private and public blockchains refereed to as "consortium blockchains" [16].

## 1.4 Smart Contracts Platforms

There is a lot of different platforms which support the creation of smart contracts. Weather it is a public blockchain platform such as Ethereum[17] or private such as Hyperledger[18]. Ethereum with its EVM (Ethereum Virtual Machine) and Solidity programing language is the most known platform bringing a standard that many other platforms, such as Hyperledger and Ubiq [19], build on. NEO [20] platform is the first blockchain platform launched in China, with its own NeoVM (NEO Virtual Machine).

## 1.5 Ethereum

Ethereum is an open-source platform for blockchain applications with its own blockchain and cryptocurrency ether. This platform offers environment to run decentralized applications based on smart contracts. As they claim "applications that run exactly as programmed without any possibility of downtime, censorship, fraud or third party interference."[17] Unlike bitcoin it offers much more than one kind of transaction, it allows users to create custom operations of any complexity and ,more importantly, save the state. It is a stateful, it can detect changes to data and remember them [21].

### 1.5.1 Ethereum Virtual Machine

The keystone of Ethereum is the Ethereum Virtual Machine (EVM), which is the byte code execution environment for the Ethereum smart contracts. To define it more closely "The EVM is a single, global 256-bit "computer" in which all transactions are local on each node of the network, and executed in

relative synchrony. It's a globally accessible virtual machine, composed of lots of smaller computers."[21] EVM is Turing-complete, therefore it can execute a byte code of discretionary algorithmic complexity. Every node in Ethereum peer-to-peer network blockchain infrastructure runs the EVM and executes the same instructions, in order to maintain consensus of the blockchain. "Decentralized consensus gives Ethereum extreme levels of fault tolerance, ensures zero downtime, and makes data stored on the blockchain forever unchangeable and censorship-resistant."[22]

### 1.5.2 Ethereum Accounts and Contracts

#### Externally Owned Accounts

Externally Owned Accounts (EOA) are accounts controlled by private keys. They have their ether balance, contain no code, can transfer ethers or trigger contract code, These accounts are usually maintained by a program called wallet (Ethereum Mist wallet).

#### Contract Accounts

Ethereum contracts are peaces of code in the EVM bytecode that live on the Ethereum blockchain. They are usually written in higher level languages, such as Solidity. "Contracts in Ethereum should not be seen as something that should be 'fulfilled' or 'complied with'; rather, they are more like 'autonomous agents' that live inside of the Ethereum execution environment, always executing a specific piece of code when 'poked' by a message or transaction"[22].

Contract has its own ether balance, associated code, own persistent storage that it can manipulate with, it can perform complex operations and call other contracts. Though, contracts should not perform non-deterministic operations, as the result must strictly be the same across all nodes, to agree on the outcome of the operation, and it should be obvious when to execute the contract to all nodes, therefore contract accounts perform operations (functions) that are activated by transactions from EOA or messages from other contracts [22].

Transactions in Ethereum are signed data packages sent by an EOA to another account and are recorded on the blockchain. Messages on the other hand can be sent between contracts and are performed locally on the EVM. The result of both transaction and message is the recipients account running its code [22].

Contract Accounts are than accounts controlled by their contract code and are created when the contract is deployed on the Ethereum blockchain.

### 1.5.3 Gas

Contract code executes in parallel on all network nodes, this is due to the main principle of blockchain to achieve a consensus that all nodes share the same code thus the result of execution is the same, without third parties needed. But such "parallel" execution is logically very expensive. Gas is an execution fee payed by the sender of a transaction for every computational step that is carried out. Every transaction sent must specify a gas limit to be spent, if the transaction execution exceeds this limit, the execution stops and all changes are reverted. Gas is usually purchased automatically by Ethereum clients for ethers [22].

Because of the costs, when creating Dapps, usually only the necessary parts are put on the blockchain and the rest of the logic is implemented outside, communicating with blockchain through RPC interface.

### 1.5.4 Communication With the Network

To communication with the blockchain network is possible either through a Ethereum client such as geth or Mist or a RPC interface, using a subset of JSON-RPC 2.0 specification, offered by an Ethereum node. The RPC interface gives possibility to access the blockchain and node functionality. Because using JSON-RPC is complicated and error prone there exists libraries such as web3.js. Through these libraries one can deploy contracts, interact with contracts or listen to contract Events. Therefore it enables creation of Dapps.

From the other point of view, a contract cannot directly return values. Contract can use events, which are data structures that when raised they are logged in the Ethereum log on the Blockchain. The frontend application can than actively read the log and 'listen' for the event to perform some action in asynchronous manner. Events and logs are mapped to contract addresses and contain data added by contract.

Chapter 2

# Evaluation of BC and EE Compatibility

In the first chapter we were describing theoretical foundations for understanding Enterprise Engineering and blockchain. Now that we understand what this methodology and technology are capable of, we can evaluate their compatibility and possible connection.

The first important thing to realize is that Enterprise Engineering is a scientific discipline with an underlying methodology of modeling processes DEMO. On the contrary, blockchain and smart contracts are a technology. But from the nature of the problems they are both addressing and even from the underlying terminology they use it seems like they could be used together. This is of course a more challenging question and one should thoroughly understand both of them to bring about the right way of using this methodology and technology together.

## 2.1 Smart Contract Misconceptions

Blockchain is undoubtedly an interesting and innovative technology with a great potential. The solutions to common problems it introduces can change variety of processes in many different areas. For example, it can be used to tackle the need of trust and prevent frauds. There have been a lot of great ideas of what could it be used for, such as suggested in a book Blockchain by Melany Swan [9]. But some of the ideas were formed on false assumptions or lack of insights into BC, therefore, some common misconceptions have been developed considering blockchain and smart contract applications. We are going to look at some of them to prevent possible misconceptions in our theory.

#### 2.1.0.1 Autonomous Smart Contracts

An idea that was not clear to me as well initially is that smart contracts can operate fully autonomously. The idea is partly true, but more in means of immutably following stated logic, rather than performing actions independently. Smart contracts are not programs, that are active all the time, they are peaces of code that are run only when invoked. In Ethereum this is, as explained in the first chapter, possible either by sending transaction or message to the contracts address. So, the idea often presented, that smart contract actively waits for some event (certain date) and than executes itself is a misconception [23].

#### 2.1.0.2 External services

One of the very common attributes we find, when researching smart contracts, is that they are designed to use external data. But this is not that easy to achieve and it is given by the very principle of determinism which is an essential feature of blockchain. When running a smart contract, all nodes must come to the same result, therefore must operate on the same data. Using external data sources to gather data for the smart contract‘s execution is impossible, as we cannot be sure that the same data will be served to all nodes. Secondly, smart contracts cannot be self-initiated.

The solution for this is rather simple. All data used must be determined at the evocation of the smart contract. Data must be sent to the contract as a parameter of evoked function or produced by a so-called oracles. Oracles are ethereum design pattern and serve as “the interface between contracts and the outside. Technically, they are just contracts, and as such their state can be updated by sending them transactions. In practice, instead of querying an external service, a contract queries an oracle; and when the external service needs to update its data, it sends a suitable transaction to the oracle.”[24] This is a common solution to the need of external data, but the fallback is that we again rely on an centralized external service, that we have to have trust in.

Furthermore, smart contracts should not initiate any action outside blockchain. For example, it might be a good idea for a smart contract to call an external API when some condition has been met. In this moment, there is at least 23880 [25] active nodes in the Ethereum network, which would all call an API with the same request. That is exactly what smart contract or any blockchain transaction should not be used for. One must understand, it is not an executional system, it is more of a notarization system or controlling system, a trustless database.

#### 2.1.0.3 Privacy issues

As blockchain is a distributed database, there is no access control to the data and actions it holds. Every node can see it all, a transparency by nature. In

Bitcoin network this is not a problem, because there is the anonymity of the addresses, that makes it unimportant that we can see the money transfers as we can not identify nor the sender, neither the receiver. On the other hand, with smart contracts we can store any data. Moreover, we probably want to recognize some addresses and know who operates behind them and restrict actions to certain addresses. Therefore, it should be considered thoroughly what we store in public blockchain end ensure security of confidential information.

## 2.2 EE and BC Compatibility

Taking into consideration everything we discussed in the previous section, we can now start to think about the proper way to connect Enterprise Engineering and BC. We have a theoretical foundation of an enterprise and a methodology of how to capture its essence and processes, on one side. On the other side, blockchain is a technology. The compatibility would be a question of using blockchain as part of the implementation of the enterprise information system, based on EE.

### 2.2.1 BC As Transaction Execution System

The first idea might be to implement the whole transaction execution, Demo engine, on blockchain through smart contracts. With all the limitations and misconceptions introduced earlier, it might not be possible to implement full business logic and, moreover, there is no need to run the exact same transaction execution multiplied on thousands of computers. Furthermore, transaction execution on blockchain is not always without expenses, this may vary based on platform used. But in general, why chose this approach when we have applications in ”regular” programming languages, which once developed are free of cost.

The second idea is to choose only some transactions, of which full or partial execution on blockchain would bring benefits. We can make use of blockchain‘s notarization of SC code and secure a trustless transaction execution when operating with untrusted third parties or multiple organizations. For example, if we have a contract that states once a certain amount of money is paid an asset will be transfered, using BC, such logic can be trustlessly implemented by smart contract, ensuring that the transaction will execute as desired. This alternative seems logical, practical and makes use of the primary benefits that BC technology brings to the table.

### 2.2.2 BC As Notarization System

Looking at how we defined blockchain in the first chapter where we stated: “It introduces decentralized, autonomous, replicated and secure database, that

based on cryptography offers trustless network without a need of intermediary", another application of BC in EE could be to serve as a notarization system. Smart contracts can offer notarization of documents, agreements and all information related to transactions, progress and results of transactions. BC could than provide a consistent and reliable source of data, facts and transaction states for all parties involved in the process.

### 2.2.3 Conclusion

To summarize, the enterprise engineering and DEMO methodology can be used to understand and analyze the essence of a process and consequently, based on that, we can decide if notarization and trustless execution of some transactions is needed, which is where BC and smart contracts can be used. Smart contract should not replace the whole executional logic but control some parts of processes and enhance the recording of facts. Blockchain should work like one part of the entire enterprise information system.

The DEMO methodology fulfills the C4 quality criteria and in its complexity offers all the information about the process needed to evaluate and create smart contracts. We already exactly know possible transactions states, their order and names. Furthermore, from the models we know what information or documents are needed in each transaction, what are the relations between them, their flow and action rules. We than only need to decide which of them need notarization or trustless execution, and then build the smart contract fully based on what we have already revealed applying the methodology.

Chapter 3

# IT System Based on EE and SC

In this chapter we are going to look at the methods of creating BC smart contracts based on DEMO methodology, we than introduce a software architecture of an enterprise informational system based on DEMO that communicates with smart contract. Finally, we look at some existing solutions of integrating business process management systems based on BPMN and BC.

## 3.1 SC based on DEMO

In the previous chapter we introduced what could be the possible usage of smart contract in EE. We have defined that there are two possible approaches, that can also be combined:

- **Notarization** of documents, agreements and all information related to transactions, progress and results of transactions.
- **Trustless execution** of transactions or part of transactions.

As mentioned before the decision whether to use SC for the process implementation is individual for every case. In general, a good use case could be to use it when operating with untrusted third parties or multiple organizations. There is, probably, no such need for notarization or trustless execution within the internal business processes, but ,naturally, the need arises when dealing with transactions on the border of the scope of interest when communicating with external actors. BC smart contract can also represent the coordination point between the internal IT system and external actors.

### 3.1.1 DEMO Transaction As Contract

On blockchain a DEMO transaction is represented as contract. Contract has its own address, internal storage, attributes, methods and is callable by either external actor or another contract. This means that it has the functionality

to represent a DEMO transaction. For mapping to corresponding DEMO transaction we use the names defined in the Transaction Product Table for contract it self and all its . The contract than encapsulates the transaction notarization or execution.

### 3.1.2 Notarization

Notarization of a DEMO transaction can be divided into two parts:

- Notarization of the transaction data and documents
- Notarization of the transaction execution

In the first case, we are looking at using the smart contract as a storage of data. To construct a smart contract carrying a transaction data we can combine information from three models: Organization Construction Diagram, Bank Contents Table, Object Fact Diagram and Action Model. From them we can retrieve which object classes are needed for the transaction and where in the transaction execution they arise, we can evaluate the changes of the objects associated with the transaction execution. Object class can be represented as internal state variable in the contract, with the corresponding name from the DEMO. Smart contract than serves as a database for the transaction.

In the second case, we want to notarize the transaction execution. We use the Complete transaction pattern fig.1.6 from where we take all possible C-Facts and add their representation in the contract. The contract than holds its current status as C-fact. For every C-Act we create a contract method that changes the contract state to the corresponding C-Fact. Every change of C-Fact issues an ethereum system-wide notification (Event), allowing external systems to keep track of their contracts. Once the transaction is completed and the P-fact was created, another event is emitted stating the P-fact.

### 3.1.3 Execution

When representing transaction execution as contract we need to consider the purpose of the transaction and its semantics. As some parts of the transactions can be performed tacitly some parts need a confirmation that can be performed automatically a and some need a explicit actor based approval. In this manner we can look at contract from two aspects:

1. “The operational aspects: these are the parts of the contract that we wish to automate, which typically derive from consideration of precise actions to be taken by the parties and therefore are concerned with performing the contract.”[26]
2. “The non-operational aspects: these are the parts of the contract that we do not wish to (or cannot) automate.”[26]

To implement the transaction execution in the contract we need to understand the whole DEMO model and the relationships between transactions. More specifically, we should focus on Organization Construction Diagram, Process Structure Diagram and Action Model, considering those are the models we will mainly use.

In the PSD model we have defined all the response links and wait links, and we can see the enclosed transactions. The action rules from AM define the exact operation for actor roles, this operation can be used to construct the execution logic. If we execute a certain C-Act in the contract we look at the action rules that contain this C-Act and we construct the corresponding method accordingly. Depending on the actor roles we define the executor of the method, we define general conditions based on the transaction pattern, such as that to perform a promise the contract must be requested. Finally, we translate the action rule pseudo code to contract code. In this case, we also need to add the notarization of the data that the transaction operates on. Method to execute a C-Act is named after the C-Act concatenated with the transaction name.

If the transaction encloses other transactions, we have to decide how to handle them. There are three possible solutions. Firstly, the sub-transaction can be implemented as another contract. The enclosing contract than stores the address of the sub-contract. This way when the action rule contains a response link for child transaction C-Act, we call the corresponding C-Act method of the child contract. The sub-transaction can also store the reference to the enclosing transaction to implement the wait links. Secondly, there must not always be a need to create separate contract for sub-transactions, we can implement the sub-transaction inside the main contract. This can be convenient if we are interested only in partial execution on BC for the sub-transactions. Finally, the last option is that we do not handle the sub-transactions at all and leave this outside of BC.

### 3.1.4 Extending the DEMO model

Using SC with DEMO is the part of the implementation of the organization. From this point of view the DEMO models should not change when using SC. SC only represent transactions and, in some cases, an actor role can be assigned to it. In both cases the underlying DEMO models are not affected.

As the implementation of contract can be derived from the DEMO models, if more deeply formalized, the creation of the contract code could be automatically produced from the DEMO models. In this case we would have to define a way to identify the transactions and types of their integration on the BC. A solution could be to introduce Transaction Blockchain Table fig. 3.1 that would map transactions to their BC implementation. As for the actor roles assigned to SC, this would be defined in the Actor Function Metrix.

| **Transaction** | **Data notarization** | **Transaction notarization** | **Execution** |
|---|---|---|---|
| T1 | List of Objects to notarize | Yes/No | List of C-Acts to execute |

Table 3.1: The Transaction Blockchain Table

The automation of SC creation could be a great benefit as it would bring a level of security. As explained in the paper by Alex Norta [27] referencing a crowdfunding project that was hacked because it contained security flaws, resulting in a $50 million loss. "The incident shows it is not enough to merely equip the protocol layer on top of a blockchain with a Turing-complete language such as Solidity to realize secure smart-contract management. Instead, we propose in this keynote paper that it is crucial to address a gap for secure smart-contract management pertaining to the currently ignored application-layer development."[27]

## 3.2 Software Architecture

In the previous section we have introduced how a smart contract based on DEMO methodology could be produced, we can now discuss the possible software architecture of an IT system based on Enterprise Engineering with the integration of blockchain.

The implementation of the IT system would consist of three parts:

- Enterprise engineering: DEMO models
- Enterprise information system (EIS): implementation and realization of the organization in an IT system based on EE, integration of all technologies
- Blockchain: implementation of the smart contracts based on EE

The architecture of an IT system integrating the enterprise information system and blockchain is illustrated in the fig. 3.1. The EIS contains mainly the BPM engine and blockchain API. The BMP engine is the transaction execution system. The blockchain API is an interface for communication between the BPM engine and blockchain. The communication with blockchain is carried out through a blockchain node. It contains a transaction processor and the blockchain database, which holds the smart contracts or blockchain logs. The DEMO models and methodology also serve as an interface for their cooperation as all these parts BPM engine, blockchain API and smart contract are based on them. They all use the transaction patterns, transaction names, facts, etc. defined by DEMO.

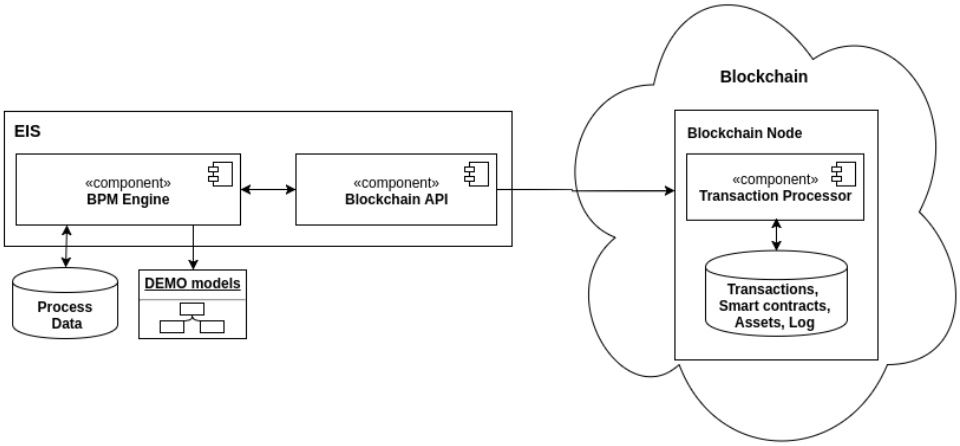



Figure 3.1: Architecture of an IT System Based on EE and BC

### EIS and BC communication

The communication between the EIS and SC is a one way interaction based on the principles described in the section 1.5.4. As BC cannot return values or call external services directly, all the interaction is handled from the EIS side.

The EIS contains an API for communicating with blockchain, such as web3.js. This API facilitates the contract deployment, sending of transactions to the contract, getting data from the contract. Using the events mechanism the API monitors the blockchain log and ”listens” for certain events. This way the API can watch the change of transaction state or results of contract execution and act on it, mostly if it is a transaction evolving external actors.

## 3.3 Comparison to Business Process Management Systems Based on BPMN

In this section we are going to look at the BPMN and some ideas of its implementation within blockchain, in order to compare it to our proposed solution.

The BPMN (Business Process Model and Notation) is a standardized notation for business process modeling. Unlike DEMO it is not a methodology but a standard maintained by the Object Management Group. The BPMN intention is to provide a support for variety of abstraction levels, from a business level to a technical implementation [28]. According to the Introduction to BPMN paper “BPMN defines a Business Process Diagram (BPD), which is based on a flowcharting technique tailored for creating graphical models of

business process operations. A Business Process Model, then, is a network of graphical objects, which are activities (i.e., work) and the flow controls that define their order of performance."[29]

### 3.3.1 SC based on BPMN

BPMN is one of the most widely used modeling standards, therefore, there have been efforts to use BPMN for smart contract implementation. One of them is described in a paper by Weber et al [6]. This paper states a similar approach to implementing a business process using BPMN on blockchain as this thesis. It recognizes two alternatives of using blockchain as "a choreography monitor, it stores the process execution"[6] or "as an active mediator among the participants, it coordinates the collaborative process execution."[6]. The approach then introduces a method of translating the BPMN model to smart contract. This method is mainly addressing collaborative process execution for participants with lack of trust.

To compare, the solution in this theses seems more flexible, it is given by the nature of DEMO models and their different approach to process modeling, where we can define more precisely which parts of the process need the use of smart contracts. The architecture based on DEMO is also focusing more on the smart contract as part of a bigger EIS than just a single process between several participants. But in general, both solutions introduce similar findings and principles about the usage of BC and process modeling and method of translating the models to smart contracts. In the end, it can come down to the comparison of DEMO methodology and BPMN standard itself and evaluating the appropriateness of their use and ability to cover all possible situations when modeling processes.

### 3.3.2 SC based on Petri nets

Another interesting solution can be found in the paper by García-Bañuelos [30]. This paper focuses on optimized execution on blockchain. It defines an method of modeling processes defined by BPMN into smart contracts through the use of optimized Petri nets. "The method takes as input a BPMN process model. The model is fist translated into a Petri net. An analysis algorithm is applied to determine, where applicable, the guards that constrain the execution of each task. Next, reduction rules are applied to the Petri net to eliminate invisible transitions and spurious places. The transitions in the reduced net are annotated with the guards gathered by the previous analysis. Finally, the reduced net is compiled into Solidity."[30]. The limitations of this approach are that it focuses on encoding the control-flow and evaluation of data conditions, but missing out parts like how the participants would be bound to the contract instance and access control. It also focuses only on a subset of BPMN notation.

Chapter 4

# Proof of Concept

In previous chapters we proposed a software architecture based on connection of DEMO methodology and blockchain smart contracts In this part we describe a proof of concept using a financial transaction, the process of mortgage, implemented in Ethereum Solidity programing language for smart contracts. The proof of concept is composed of

- **Process description** - contains description of the mortgage process and describes the possible benefits
- **DEMO models** - analyses mortgage process and DEMO models of the mortgage process based on its description
- **DEMO and Smart Contract** - connection of DEMO and smart contract and implementation of smart contract in Ethereum Solidity based on DEMO methodology.

Please take into consideration that this implementation serves as proof of concept, there is need of more extensive research and testing on real blockchain. This proof of concept also makes use hypotheses or expectations regarding blockchain technology such as digital identity.

## 4.1 Technologies used

#### 4.1.0.1 Solidity

Solidity [31] is a programming language to implement smart contracts specially designed for the Ethereum Virtual Machine (EVM). It is a Turing-complete high-level language compiled to the EVM bytecode.

Solidity was chosen because it is developed under Ethereum and is the most used language for smart contracts for EVM, although there are some other languages, Solidity is the most developed language amongst them.

The building block in solidity is a contract which is similar to class in object-oriented programming. Contract contains persistent data in state variables, functions to operate on this data and it also supports inheritance. Contract can further contain function modifiers, events, struct types and other structures to allow implementation of complex contracts and full usage of EVM and blockchain capabilities.

A smart contract written in solidity can be created either through a ethereum transaction or by another already running contract, just like we would create an instance of a class. Either way the contract code is than compiled to the EVM bytecode, new transaction is created holding the code and deployed to blockchain, returning the address of the contract for further interaction.

Solidity contracts can also be created, deployed and interacted with programatically using the JavaScript API web3.js, which is an ethereum compatible library implementing the Generic JSON RPC spec, that provides a convenient interface for communication with Ethereum nodes [32].

#### 4.1.0.2 Remix

Remix is a browser-based IDE for creating smart contracts with integrated debugging and testing environment. Remix offers development, compilation and deployment of solidity contracts as well as access to already deployed contracts. The testing environment allows running the transactions in a sandbox blockchain in the browser with JavaScript VM with a possibility to switch between virtual accounts and spend virtual ethers for full smart contract testing [33].

## 4.2 Process description

In this part the financial transaction is described and discussed to evaluate possible benefits of modeling this process with DEMO and implementing with smart contract. The chosen financial transaction is the process of mortgage, as it appears as a good candidate to demonstrate the advantages and compatibility of DEMO and smart contracts.

The mortgage contract is a rather confusing and complicated process involving several parties, dependent processes, level of trust between parties and a lot of documents proving results of auxiliary processes and above all notarization. This aspects all contribute to overall complexity and costs of the process. Thus it appears as a good use case where modeling by DEMO would capture the essence of the process and smart contract could offer a automated notarization, data sharing between parties and payment processing reducing the need of manual processes as illustrated by 4.1.

The description of the mortgage process is based on research made of several on-line mortgage guides [34] [35] and consultation with real estate agent. The description of the process was than modeled by flow chart to fully

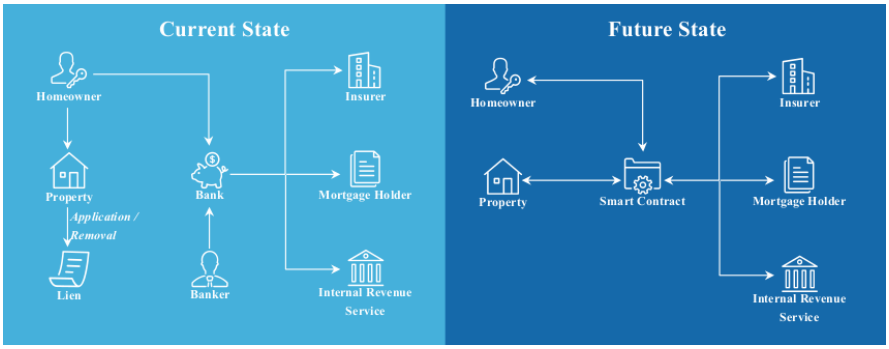



Figure 4.1: Mortgage process changed using smart contract [5]

understand and illustrate the process. For needed illustration we omit the common pre-approval faze.

**Mortgage process description**

1. **step** - Buyer applies for mortgage in bank. He fills in an application with basic information (personal data, employment, income, marital status, etc.). Based on primary application bank requests further documents, needed for full client screening (credit report, account statements, income certificates/invoices, etc.) and documents for desired property such as reservation contract, property appraisal, draft of sales contract. When the bank internally approves the mortgage, they prepare the Mortgage contract which states all the details such as the finance charge, annual percentage rate (APR), number of payments you will make, amount of each payment (for fixed-rate loans), late payment charges that may apply and total amount you will pay in principal and finance charges over the life of the loan and further conditions to be met to issue the funds such as property insurance and states the closing date.

2. **step** - Once the mortgage is approved by bank and closing is scheduled, client or bank secures a homeowner insurance for property.

3. **step** - At the closing, all of the needed parties (loaner, buyer, seller) meet at a notary. The sale contract, the mortgage contract and documents for cadastre of real estates are signed. After the documents for cadastre and lien are registered at the cadaster, the bank issues the funds and the deal is finalized.

4. **step** - The loan servicing, the steps taken to maintain a loan from the time it is closed until it is paid off, then starts. Client is obligated to pay monthly pay-off payments. Once the mortgage is fully paid-off the bank releases the property lien and client becomes a rightful owner of the property.

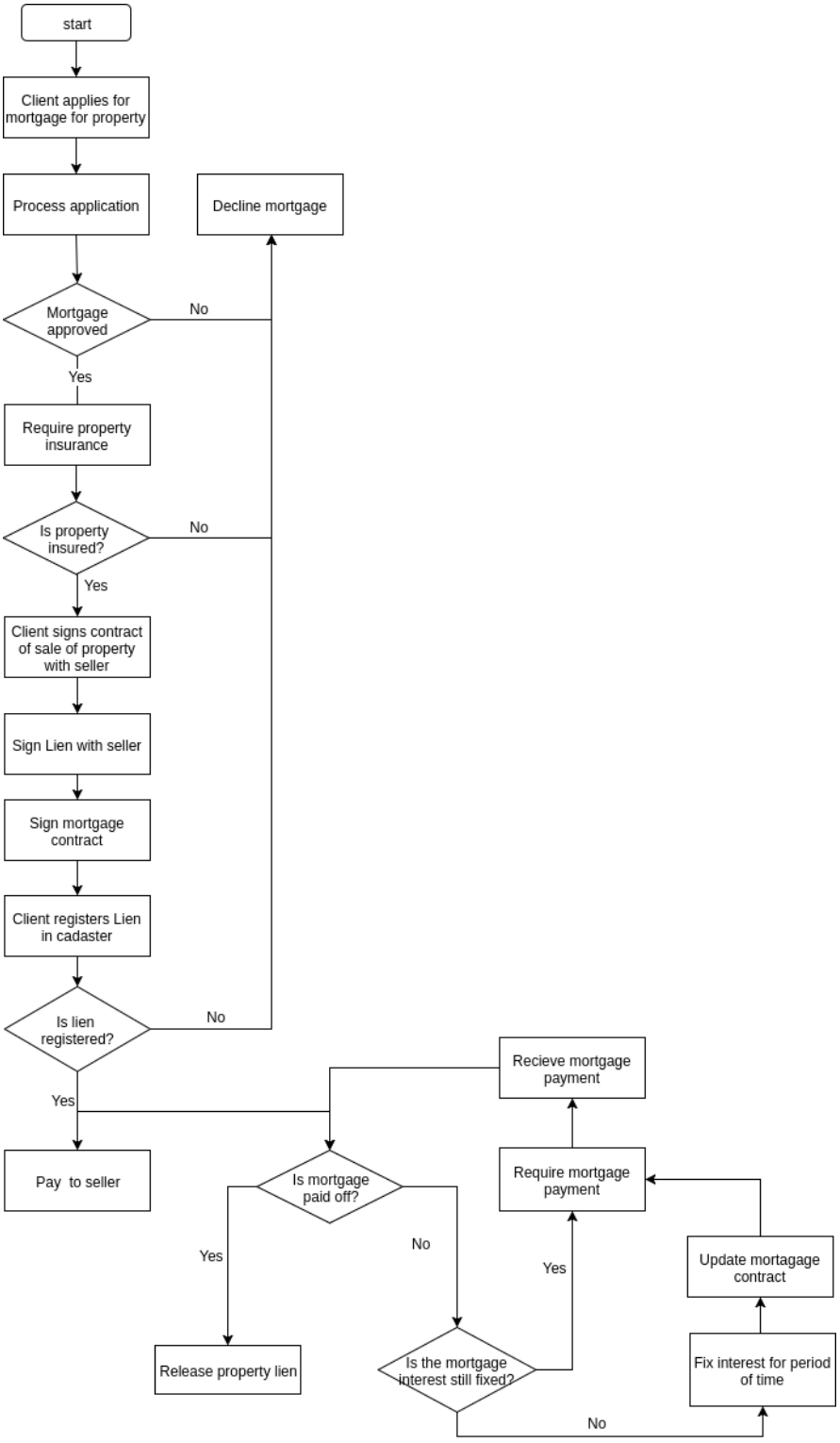



Figure 4.2: Mortgage process flow chart

## 4.3 DEMO Model

Once the mortgage process is described and conceived we can start to reveal the essence of the process and model it using DEMO methodology. We first analyze the process and than create all needed models according to the proposed method in section chapter 3.

### Mortgage process analysis

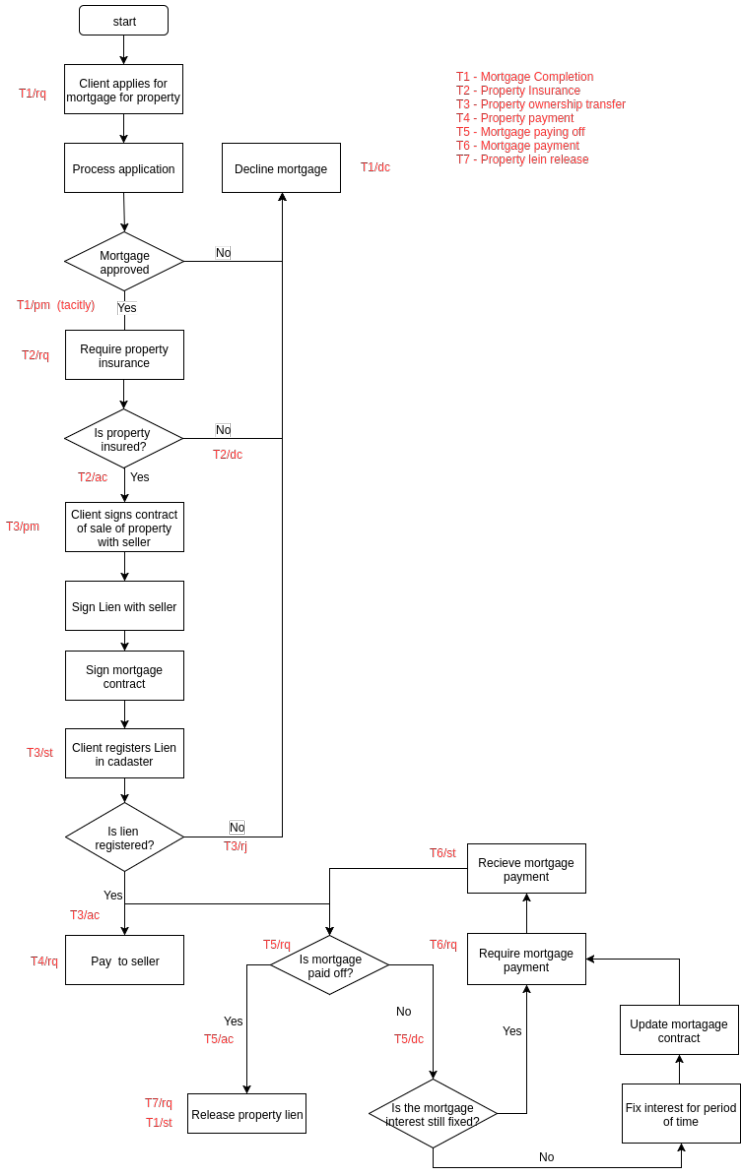



Figure 4.3: Mortgage process flow chart with identified C-acts

The most important part of revealing the essence of a process is to identify its core transactions and distinguish documental, informational and original acts. We will use the flow chart to analyze the process, for illustration we only mark out C-acts fig. 4.3

As we can see in fig. 4.3 there were seven different transactions identified with a transaction Mortgage completion representing the main process covering up all other transactions.

### Construction model

Based on the mortgage process analysis and identified transactions we can produce the Transaction Product Table tab. 4.1, Bank content table tab. 4.2 and Actor Transaction Diagram fig. 4.4.

| **Transaction kind** | **Production type** |
|---|---|
| T1 Mortgage completion | P1 Mortgage is completed |
| T2 Property insurance | P2 Property is insured |
| T3 Property ownership transfer | P3 Property ownership is transferred |
| T4 Property payment | P4 Property is paid |
| T5 Mortgage paying off | P5 Mortgage is paid off |
| T6 Mortgage payment | P6 Mortgage is paid |
| T7 Property lien release | P7 Property lien is released |

Table 4.1: The Transaction Production Table

| **Bank** | **Independent/Dependent facts** |
|---|---|
| T1 Mortgages completion | MORTGAGE<br>Mortgage is completed<br>The receiver of Mortgage<br>The property of Mortgage<br>The amount of Mortgage<br>The annual percentage rate of Mortgage<br>The final amount to pay off for Mortgage<br>The number of payments for Mortgage<br>The amount of payment for Mortgage |

| | |
|---|---|
| AT1 Client facts | CLIENT<br>personal data of Client<br>income of Client<br>employment information of Client<br>marital status of Client<br>credit report of Client<br>account statements of Client<br>Income certificates of Client |
| T2 Property Insurance | PROPERTY<br>Property is insured |
| T3 Property ownership transfer | PROPERTY<br>Property ownership is transferred<br>The owner of the Property<br>The lien of the Property<br>The amount to pay for Property |
| T4 Property payment | PROPERTY<br>Property is paid<br>The amount paid for Property |
| T5 Mortgage paying off | MORTGAGE<br>Mortgage is paid off<br>The amount paid off for Mortgage |
| T6 Mortgage payment | MORTGAGE<br>Mortgage is paid<br>The amount paid for Mortgage |

Table 4.2: The Bank Contents Table

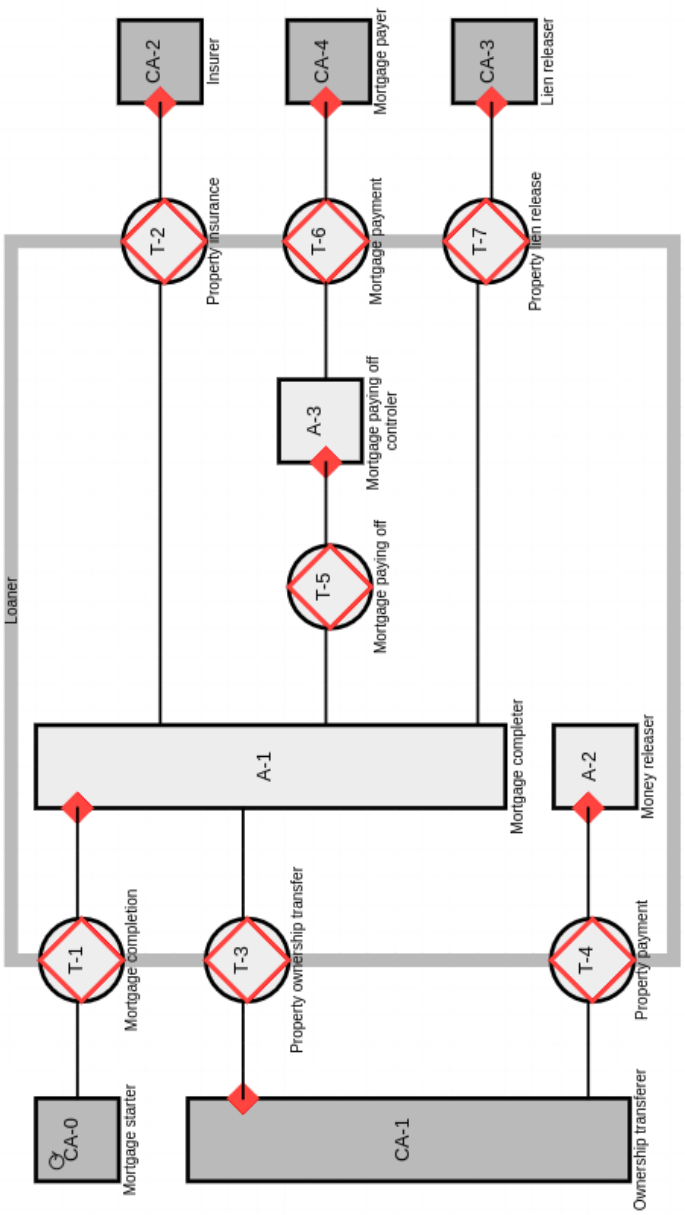



Figure 4.4: The Actor Transaction Diagram

### Process model

To fully understand the transactions flow we use the Process Structure Diagram fig. 4.5 that illustrates the dependencies between transaction‘s states. This conditions for the order of transaction‘s state are important when implementing the smart contract.

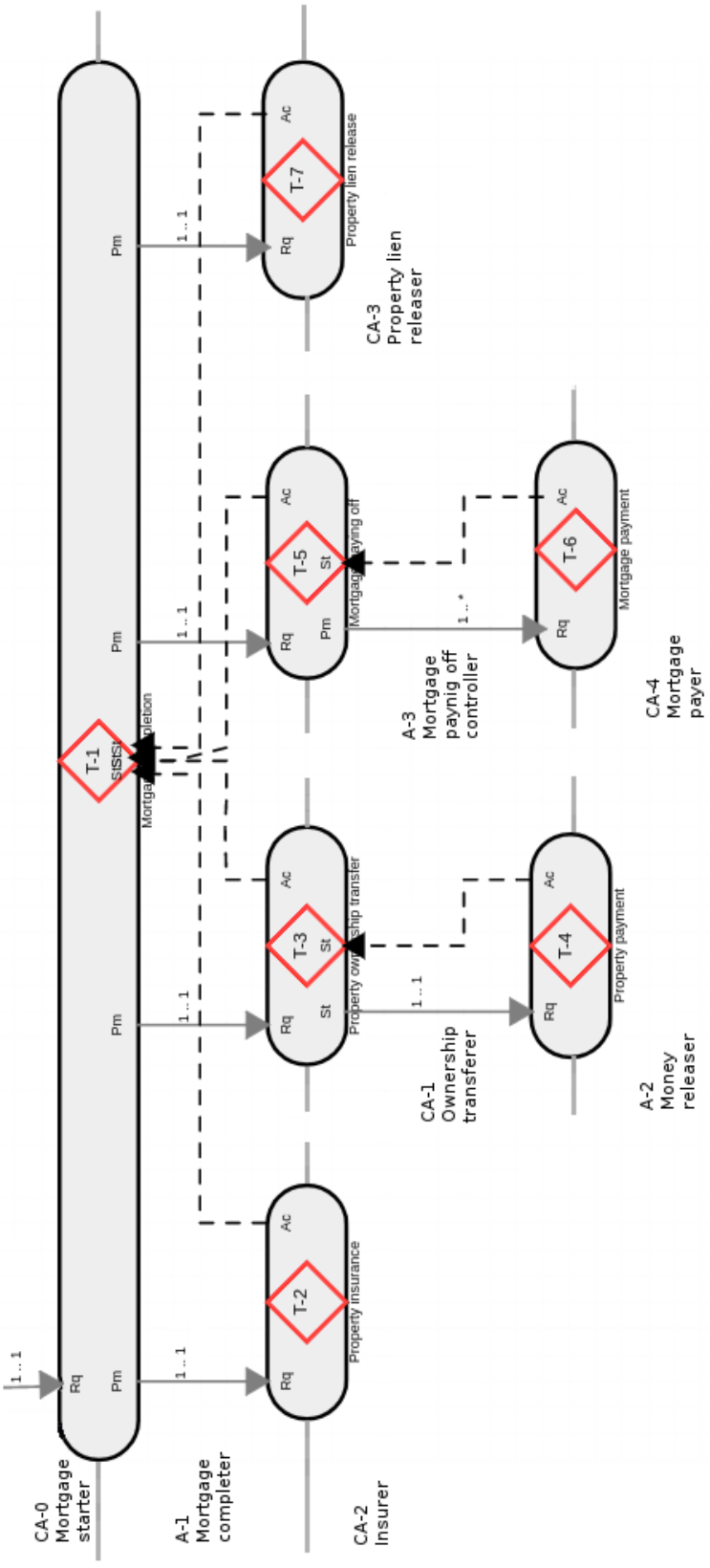





Figure 4.5: The Process Structure Diagram

### Fact model

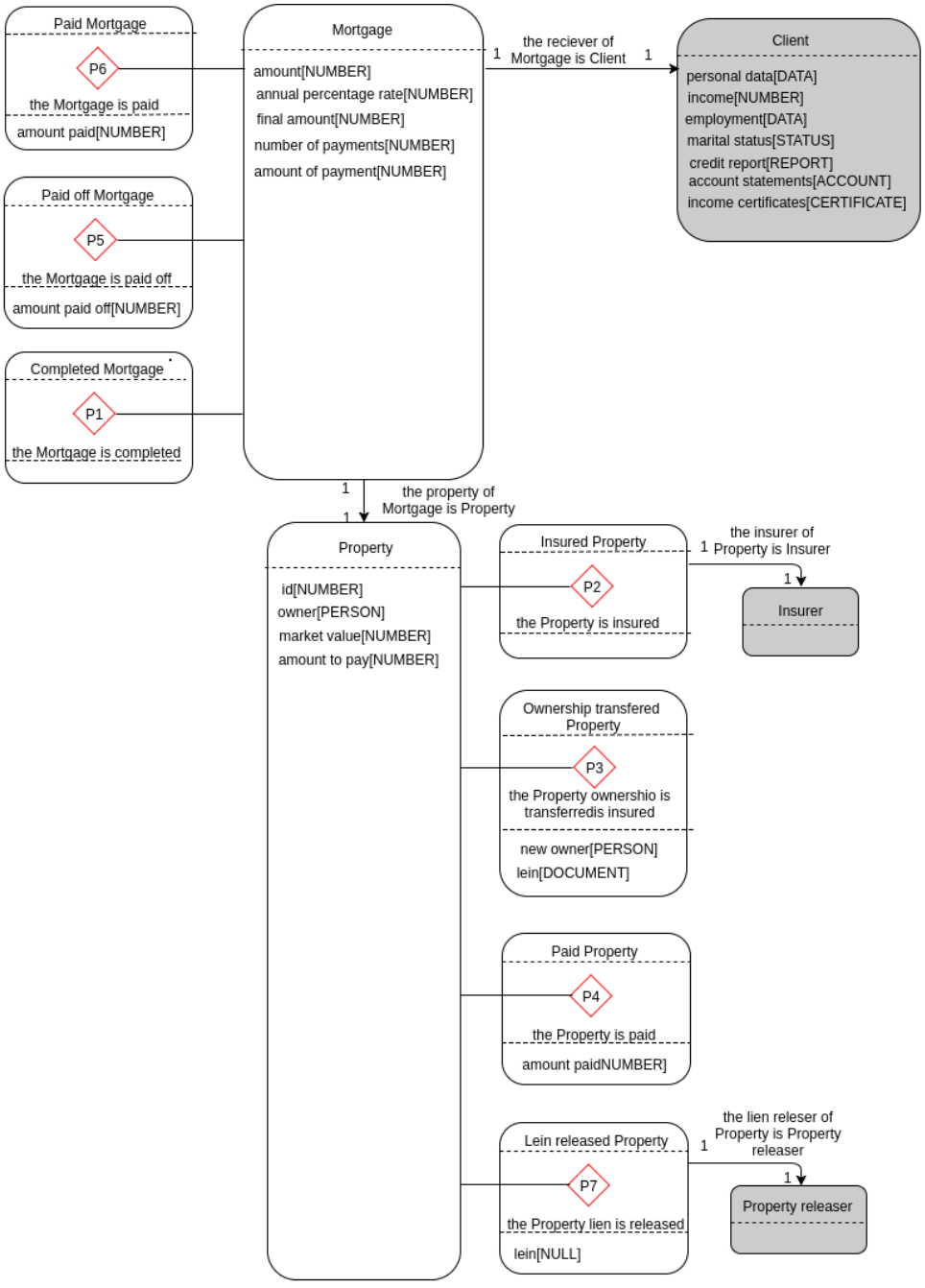



Figure 4.6: The Object Fact Diagram

### Action model

The Action rules are a final DEMO model and is the one most needed for smart contract creation. The structure and content of the smart contract can be directly mapped to each action rule. By creating the action rules we basically create the logic on which the smart contract operates.

| | |
|---|---|
| when | mortgage completion for new Mortgage is requested (T1/rq)<br>with the receiver of Mortgage is Client<br>the property of Mortgage is Property |
| assess | justice: the performer of the request is the receiver of Mortgage<br>sincerity: <no specific conditions><br>truth: Mortgage for property for receiver is approved |
| if | complying with request is considered justifiable |
| then | promise mortgage completion of Mortgage [T1/pm] |
| else | decline mortgage completion for Mortgage [T1/dc] |

Table 4.3: The Action Rule 1

| | |
|---|---|
| when | mortgage completion for Mortgage is promised (T1/pm) |
| assess | justice: the performer of the promise is the mortgage completer of Mortgage<br>sincerity: <no specific conditions><br>truth: <no specific conditions> |
| if | complying with request is considered justifiable |
| then | request property insurance of Property [T2/rq]<br>with the property to insure is the property of Mortgage<br><br>request property ownership transfer of Property [T3/rq]<br>with the property to transfer ownership is the property of Mortgage<br>the new owner to be of property is client of Mortgage<br>the lien of the property will be for the loaner of Mortgage<br>the amount to release to pay for property is the amount of Mortgage |

Table 4.4: The Action Rule 2

| | |
|---|---|
| when | property insurance for Property is stated (T2/pm) |
| assess | justice: the performer of the state is the insurer of Mortgage<br>sincerity: <no specific conditions ><br>truth: The property insured is the property of Mortgage |
| if | complying with request is considered justifiable |
| then | accept property insurance of Property [T2/ac] |
| else | reject property insurance of Property [T2/rj] |

Table 4.5: The Action Rule 3

| | |
|---|---|
| when | property ownership transfer is stated (T3/st) |
| assess | justice: the performer of the state is the ownership transferer of Property<br>sincerity: <no specific conditions ><br>truth: the property of transferred ownership is the property of Mortgage<br>the new owner of property is client of Mortgage<br>the lien of the property is for the loaner of Mortgage |
| if | complying with request is considered justifiable |
| then | accept property ownership transfer of Property [T3/ac] |
| else | reject property ownership transfer of Property [T3/rj] |

Table 4.6: The Action Rule 4

| | |
|---|---|
| when | mortgage completion for Mortgage is promised (T1/pm)<br>while property insurance is accepted<br>property ownership transfer is accepted |
| assess | justice: the performer of the promise is the mortgage completer of Mortgage<br>sincerity: <no specific conditions ><br>truth: textless no specific conditions ><br>the new owner of property is client of Mortgage<br>the lien of the property is for the loaner of Mortgage<br>the amount released to pay for property is the amount of Mortgage |

| | |
|---|---|
| if | complying with request is considered justifiable |
| then | request mortgage paying off for Mortgage [T5/rq]<br>with the amount to pay off is the amount to pay off of Mortgage |

Table 4.7: The Action Rule 5

| | |
|---|---|
| when | mortgage completion for Mortgage is promised (T1/pm)<br>while property insurance is accepted<br>property ownership transfer is accepted |
| assess | justice: the performer of the promise is the mortgage completer of Mortgage<br>sincerity: <no specific conditions ><br>truth: the new owner of property is client of Mortgage<br>the lien of the property is for the loaner of Mortgage<br>the amount released to pay for property is the amount of Mortgage |
| if | complying with request is considered justifiable |
| then | request mortgage paying off for Mortgage [T5/rq]<br>with the amount to pay off is the amount to pay off of Mortgage |

Table 4.8: The Action Rule 6

| | |
|---|---|
| when | mortgage paying off for Mortgage is requested (T5/rq)<br>with the amount to pay off equals to the amount to pay off for Mortgage |
| assess | justice: the performer of the request is mortgage completer of Mortgage<br>sincerity: <no specific conditions ><br>truth: the amount to pay off is equal the amount to pay off of Mortgage |
| if | complying with request is considered justifiable |
| then | promise mortgage completion of Mortgage [T5/pm] |
| else | decline mortgage completion for Mortgage [T5/dc] |

Table 4.9: The Action Rule 7

| | |
|---|---|
| when | mortgage paying off for Mortgage is promised (T5/pm) |
| assess | justice: the performer of the promise is the mortgage paying off controller of Mortgage<br>sincerity: <no specific conditions ><br>truth: <no specific conditions > |
| if | complying with request is considered justifiable |
| then | request mortgage payment for Mortgage [T6/rq]<br>with the amount to pay for Mortgage is equal to the amount of each payment for Mortgage |

Table 4.10: The Action Rule 8

| | |
|---|---|
| when | mortgage payment for Mortgage is stated (T6/st)<br>with the amount paid for Mortgage is some Money |
| assess | justice: the performer of the state is the payer for Mortgage<br>sincerity: <no specific conditions ><br>truth: with the amount paid for Mortgage is equal to the amount of each payment for Mortgage |
| if | complying with request is considered justifiable |
| then | accept mortgage payment for Mortgage [T6/ac] |
| else | reject mortgage payment for Mortgage [T6/rj] |

Table 4.11: The Action Rule 9

| | |
|---|---|
| when | mortgage paying off for Mortgage is promised (T5/pm)<br>while mortgage payment for Mortgage is accepted |
| assess | justice: the performer of the promise is the mortgage paying off controller of Mortgage<br>sincerity: <no specific conditions ><br>truth: The summary of amount of mortgage payments for Mortgage is equal to the the amount to pay off for Mortgage |
| if | complying with request is considered justifiable |
| then | execute mortgage paying off for Mortgage [T5/ex]<br>state mortgage paying off for Mortgage [T5/st] |

| | |
|---|---|
| else | request mortgage payment for Mortgage [T6/rq]<br>with the amount to pay for Mortgage is equal to the amount of each payment for Mortgage |

Table 4.12: The Action Rule 10

| | |
|---|---|
| when | mortgage paying off for Mortgage is stated(T5/st) |
| assess | justice: the performer of the state is the mortgage paying off controller of Mortgage<br>sincerity: <no specific conditions ><br>truth: the amount paid off is equal to the amount to pay off for Mortgage |
| if | complying with request is considered justifiable |
| then | accept mortgage paying off for Mortgage [T3/ac]<br>request property lien release of Property [T7/rq]<br>with the property to release lien of is the property of Mortgage<br>the lien releaser is the some Property Lien Releaser |
| else | reject mortgage paying off for Mortgage [T3/rj] |

Table 4.13: The Action Rule 11

| | |
|---|---|
| when | property lien release of Property is stated (T7/st) |
| assess | justice: the performer of the state is the property lien releaser of property<br>sincerity: <no specific conditions ><br>truth: the property to be released is equal to the property of Mortgage<br>The lien released is the lien of loaner of Mortgage |
| if | complying with request is considered justifiable |
| then | accept property lien release of Property [T7/ac]<br>execute mortgage completion of Mortgage [T1/ex]<br>state mortgage completion of Mortgage [T1/st] |
| else | reject property lien release of Property [T7/rj] |

Table 4.14: The Action Rule 12

## 4.4 DEMO and Smart Contract

Analyzing the process through DEMO models we can now evaluate the possible use of smart contract.

For illustration we will assign actors to the actor roles as follows:

- Loaner: A-1, A-2
- Client: CA-0, CA-1, CA-4
- Insurer: CA-2
- Property Releaser: CA-2

For illustration of possible benefits of blockchain the implementation relies on these pre-requisites:

- Implementation of digital identity
- Adaption of public key infrastructure between actors
- Possibility of mortgage payment in cryptocurrency

Here is what smart contracts based on DEMO models described by this theses look like. For clear connection to the DEMO methodology we use terms from the DEMO model. Full solidity code can be found on the enclosed CD.

### 4.4.1 Generic Transaction as Smart Contract

We created a generic Transaction contract that serves as a parent class for contracts based on transactions. It holds some general functionality such as common state variables: name, product fact, current transaction fact, initiator and executor.

```
contract Transaction {

  enum C_facts { Initial, Requested, Promised, Declined, Stated,
  Accepted, Rejected }

  string name;
  string P_fact;
  C_facts public current_c_fact;
  address public initiator;
  address public executor;
```

The contract defines access modifiers to authorize the actor evoking the function call or simple coordination facts control. The authorization pattern is a common design pattern for Ethereum smart contracts to restrict code

execution to only certain address according to the caller address accessible in `msg.sender` [24].

```
modifier onlyExecutor {
  require(msg.sender == executor);
  _;
}

modifier isRequested {
  require (current_c_fact == C_facts.Requested);
  _;
}
```

Further it contains functions to represent coordination acts of the standard transaction pattern. Here we use Solidity events, which allow libraries such as web3.js to listen to changes in transaction's facts and also to log the new coordination facts and thus keep the history of transaction states.

```
event NewFact(
  C_facts c_fact,
  string transaction_name
);

function request() internal {
  current_c_fact = C_facts.Requested;
  NewFact(C_facts.Requested, name);
}
```

### 4.4.2 Mortgage Completion as Smart Contract

We have 7 consequent transactions with the Mortgage completion being the parent transaction. The Mortgage completion transaction can be looked at as the contract between the Loaner and the Client with conditions needed to be fulfilled for the contract to be finished. This contract‘s underlying transaction needs notarization, data sharing and trustless control of the execution. So first transaction where we can use smart contract is Mortgage completion. The DEMO model itself would not change.

The idea is that this smart contract would be deployed to blockchain by the Loaner of the Mortgage, after the T1-Mortgage completion is requested.

We created a contract called `MortgageCompletion`. The state variables Mortgage and Property based on the BCT (fig. 4.2) and OFD (fig. 4.6) models represent a data needed for the contract. Further we declare that the contract has four sub-transactions its state depends on according to the PSD

model (fig. 4.5). Three of them are represented by SubTransaction structure and one is represented by another contract, which will be discussed later. Finally we declare the addresses of the actors for the sub-transactions, for control of access and conditions control.

```
contract MortgageCompletion is Transaction {

  struct Property {
    string id;
    uint value;
    address owner;
    address lien;
    bool insured;
  }

  struct Mortgage {
    uint amount;
    uint annual_percentage_rate;
    uint final_amount;
    uint amount_of_payment;
  }

  Mortgage public mortgage;
  Property public property;

  SubTransaction public propertyInsurance = SubTransaction(
   "Property Insurance", C_facts.Initial);
  SubTransaction public propertyOwnershipTransfer = SubTransaction(
   "Property Ownership Transfer", C_facts.Initial);
  SubTransaction public propertyLeinRelease = SubTransaction(
   "Property Lein Release", C_facts.Initial);
  MortgagePaingOff public mortgagePaingOff;

  address client;
  address insurer;
  address property_releaser;
```

In the constructor of `MortgageCompletion` we declare the state variables that are already known and we define the actors addresses which are definite. (Please note that the address used are only for illustration).

```
function MortgageCompletion() Transaction("Mortgage completion",
         "Mortgage is completed",
         0x014723a09acff6d2a60dcdf7aa4aff308fddc160c,
         msg.sender)
```

```
{
  client = initiator;
  insurer = 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c;
  property_releaser = 0x14723a09acff6d2a60dcdf7aa4aff308fddc160c;
  property = Property("1",
             0xdd870fa1b7c4700f2bd7f44238821c26f7392148,
             0x0, 1000000, false);
}
```

The last part are the transaction execution functions. The logic of each step of a transaction is based on the Action rules and data changes based on BCT (fig. 4.2) and OFD (fig. 4.6) models.

The function `promiseMortgageCompletion` represents the promise of Mortgage completion based on the Action Rule 1 (tab. 4.3) and the Action Rule 2 (tab. 4.4). Here the function also performs implicit transaction execution by requesting Property insurance and Property ownership transfer.

```
function promiseMortgageCompletion(uint amount,
         uint _annual_percentage_rate,
         uint _final_amount, uint _amount_of_payment)
         isRequested onlyExecutor
{
  promise();
  mortgage.amount = amount;
  mortgage.annual_percentage_rate = _annual_percentage_rate;
  mortgage.final_amount = _final_amount;
  mortgage.amount_of_payment = _amount_of_payment;
  requestPropertyInsurance();
  requestPropertyOwnershipTransfer();
}
```

Here we have two functions that represent the Mortgage ownership transfer state and accept based on the Action Rule 4 (tab. 4.6). Here the performer of the request is not considered an authority and the conditions defined in claim to truth cannot be checked automatically, so both state and accept must be explicit. Once the accept is performed therefore the conditions from claim to truth have been verified outside smart contract, the state variables are set to desired values defined by the action rule.

```
function statePropertyOwnershipTransfer() {
  require (msg.sender == client );
  stateSub(propertyOwnershipTransfer);
}
```

```
function acceptPropertyOwnershipTransfer() onlyExecutor {
  require( propertyOwnershipTransfer.current_c_fact ==
           C_facts.Stated );
  property.owner = client;
  property.lien = this;
  acceptSub(propertyOwnershipTransfer);
}
```

On the other hand the property releaser is considered an authority so when stating the Property lien release, based on Action Rule 12 (tab. 4.14), we can perform accept `acceptSub(propertyLeinRelease)` and `state` tacitly.

```
function statePropertyLeinRelease(string _property_id)  {
  require( propertyLeinRelease.current_c_fact ==
           C_facts.Requested );
  require (msg.sender == property_releaser );
  require( keccak256(_property_id) == keccak256(property.id));
  property.lien = 0x0;
  acceptSub(propertyLeinRelease);
  state();
}
```

As stated before the transaction Mortgage Paying off is represented by another smart contract, which is created from within the `MortgageContract`.

```
function requestMortgagePaingOff() onlyExecutor returns (address) {
  require( propertyInsurance.current_c_fact == C_facts.Accepted);
  require( propertyOwnershipTransfer.current_c_fact ==
           C_facts.Accepted );
  mortgagePaingOff = new MortgagePaingOff(this, mortgage.final_amount,
                     mortgage.amount_of_payment);
  return address(mortgagePaingOff);
}
```

### 4.4.3 Mortgage Paying Off as Smart Contract

Mortgage paying off is another transaction that can be represented by smart contract. The difference is that here it is not just the transaction but also the actor A-3. This is given by the trustless payments in ethers on Ethereum blockchain and its usage in smart contract, all the executional logic applied by actor A-3 is then fully replaceable by smart contract. The DEMO model itself would not change, only the actor role of A-3 is assigned to smart contract. The initiator of this transaction is than the Mortgage completion smart contract and executor is the contract itself.

An important function is the `stateMortgagePayment` based on the Action Rule 9 (tab. 4.11) and 10 (tab. 4.12). A withdrawal pattern, the recommended method by solidity to control funds transfer, was used here [36].

```
function stateMortgagePayment() payable {
  require( mortgagePayment.current_c_fact == C_facts.Requested );
  require( msg.value == amount_of_payment);
  acceptSub(mortgagePayment);
  amount_paid += msg.value;
  pendingWithdrawal += msg.value;

  if ( amount_paid == amount ) {
    state();
    mortgageCompletion.acceptPropertyPaingOff();
  } else {
   requestMortgagePayment();
  }
}
```

## 4.5 Simulation

In this section we are going to simulate the execution of the Mortgage Completion contract using Remix, to illustrate the behavior of blockchain and the contracts.

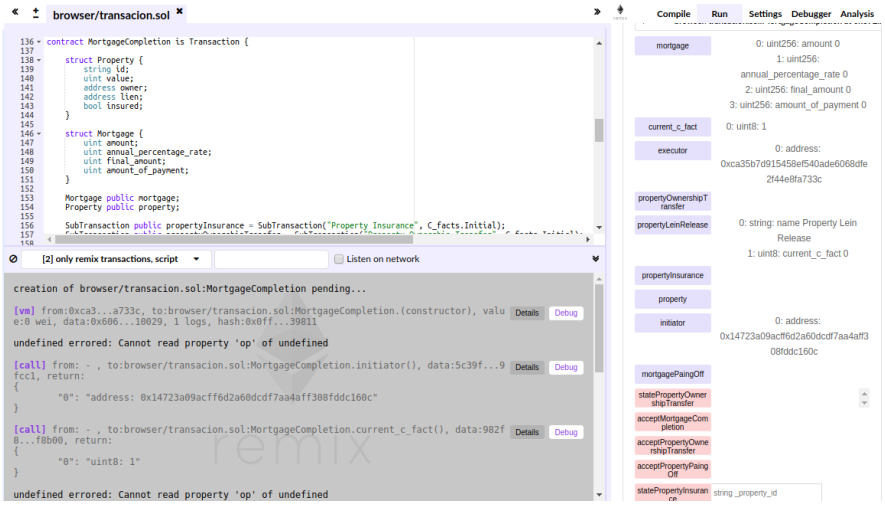



Figure 4.7: Deployment of the contract

The first step is to deploy the contract to our virtual blockchain. We choose the address to deploy from, we set the gas limit and than create the contract

MortgageCompletion. As we can see in fig 4.7 the contract was deployed, its public state variables and methods are listed on the right side. The message "undefined errored: Cannot read property 'op' of undefined" is a Remix bug, the contract deployment was successful. This step is taken once the Mortgage Completion has been requested. The contract is deployed by the loaner.

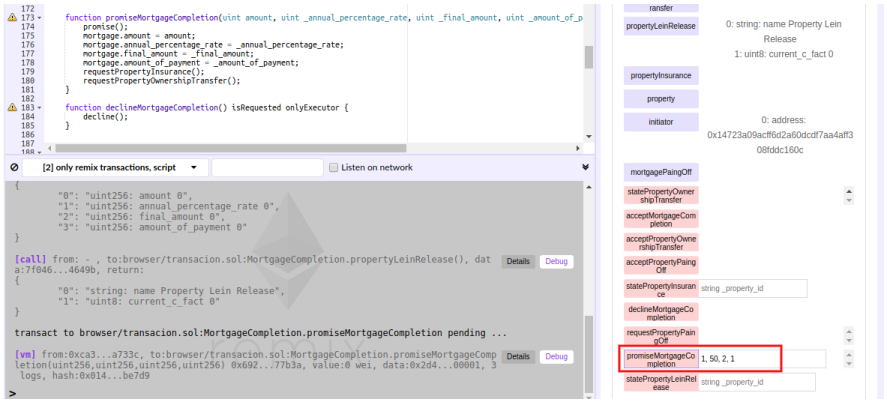


Figure 4.8: Promising the Mortgage Completion

When the request has been internally evaluated by the loaner and the conditions were finalized, the Mortgage Completion is either declined or promised. In the case of promise, data about the mortgage are sent in a blockchain transaction. For illustration we define the mortgage amount to 1 wei, with the annual percentage rate 50% and resulting on final amount to pay of 2 wei fig. 4.8. Tacit request of Property Insurance and Property Ownership Transfer is executed.

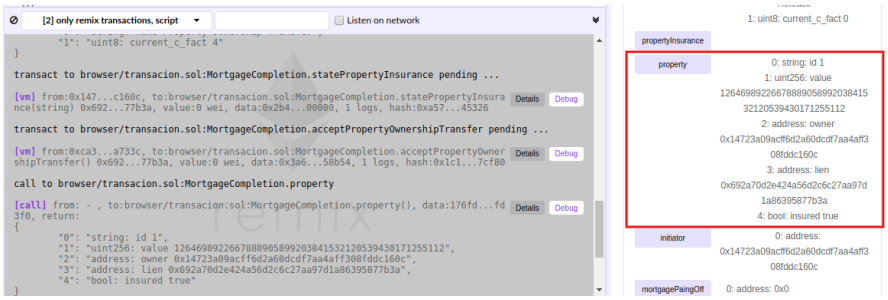


Figure 4.9: Change of state of Property

In the next step, the Property Insurance and Property Ownership Transfer are executed. To state these transactions successfully, the blockchain transaction must be sent from the address assigned to insurer and property transferer. As the insurer is considered an authority in our scenario, the accept of the Property Insurance is tacit. On the contrary, accept of the Property Ownership Transfer must be performed explicitly. When accepted the state variable

Property is updated and the address of the client is added as owner and address of the Mortgage Contract as the lien address fig. 4.9.

When both sub-transactions are accepted, the loaner can request the Mortgage Paying Off. If the loaner would try to request it sooner the blockchain transaction would not be successful and contract would be reverted fig. 4.10. In this step the Mortgage Paying off contract is created and deployed. We can get the contract's address from the main contract state variable and load it using the address. The next steps are then performed by the Mortgage Paying Off contract.

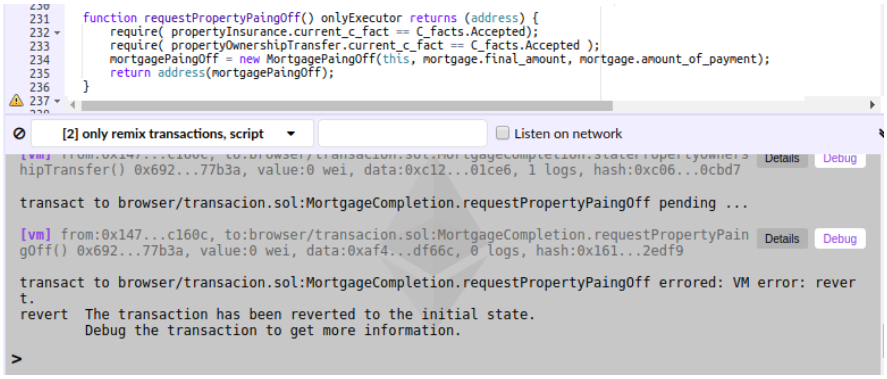



Figure 4.10: Reverting transaction

The Mortgage Paying Off contract covers the corresponding transaction and also acts like the actor A-3, because payments on blockchain can be handled automatically. When the Mortgage Paying Off is created, the final amount and amount of one payment are initialized from the Mortgage Completion contract, we define the address of the Loaner to pay the amount to and, finally, its sub-transaction Mortgage Payment is requested fig. 4.11

The mortgage payment can be sent from any address. In our scenario two payments with the amount of 1 wei must be paid. In order to accept the payment, the Mortgage payment must be requested and the sent value must be equal to the amount of one payment, otherwise the blockchain transaction is reverted and ethers are not transfered. To pay the payment we send a blockchain transaction with the desired value to the contract fig. 4.12.

Once the full amount has been paid, the Mortgage Paying Off is stated and automatically calls Mortgage Completion contract to accepted it. The Mortgage Lien Release is then requested tacitly. The Lien Releaser is considered an authority, therefore when Mortgage Lien Release is stated the accept is performed automatically, as well as the Mortgage Completion stating. As we can see in fig. 4.13 from the console detail of the blockchain two events were logged. One for accepting (5) of the Property Lien Release and one for stating (4) the Mortgage Completion. On the right side we can see that the state variable Property has been updated and the lien address was set to empty

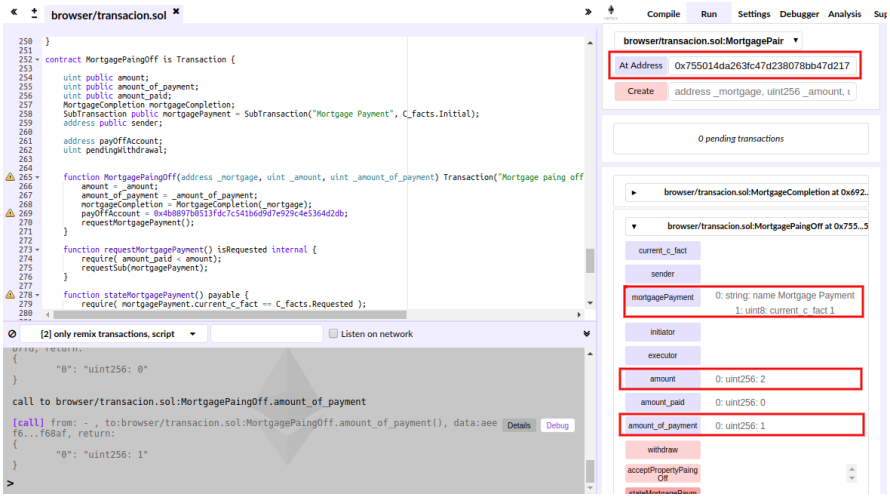


Figure 4.11: Mortgage Paying Off initialization

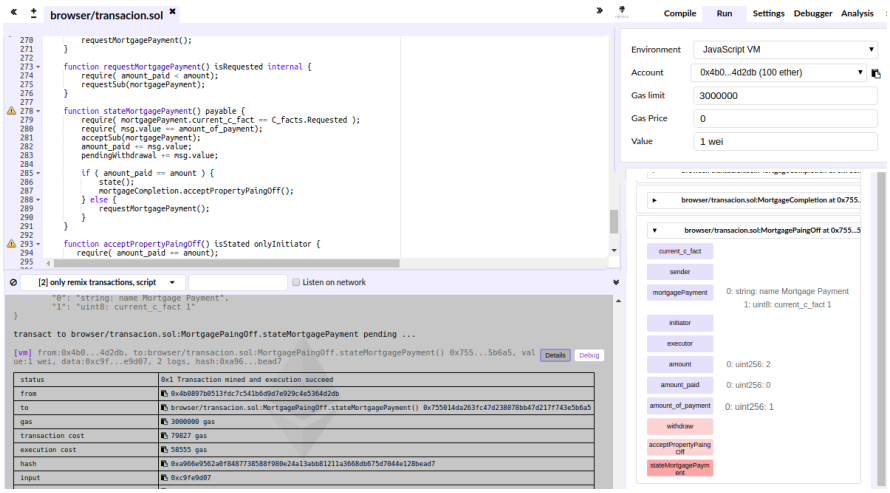


Figure 4.12: Mortgage Payment blockchain transaction

address.

Now the final step is for the client to accept the Mortgage Completion and the transaction is finished.

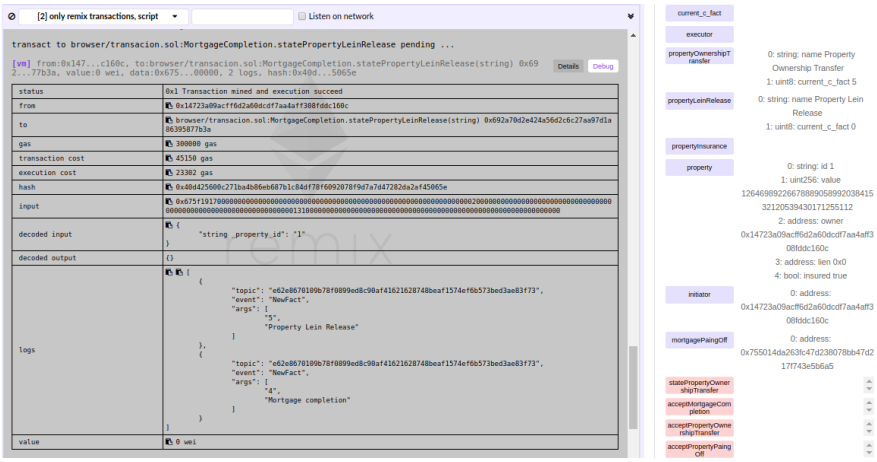



Figure 4.13: State of Lien Release

### 4.5.1 Summary

We have used the DEMO methodology to understand and reveal the essence of a mortgage process. Using the proposed principles of creating contract introduced in chapter 3 we were able to create working smart contracts based on the DEMO models.

We created a trustless notarization of the process. The contract holds immutably the agreed mortgage conditions such as amount of payment and interest rate. Further it controls execution of some parts, such as automatic mortgage payment control and automatic lien release request. This way the Client can be sure that once the mortgage has been paid off the lien will be released. It also defines a single point of access to the data and coordination for all parties as well as simplifies some steps as automatic control can be performed, allowing us to carry out some acts tacitly. Using smart contract would not change the DEMO model or the essence of the process it could help to simplify the current implementation. For example the Client would not have to bring the confirmation about insurance to the Loaner, because this is done by the smart contract, this reduces the overall process steps behind mortgage and eliminates possible bureaucracy.

# Conclusion

The main aim of this thesis was to review the capabilities of blockchain and smart contracts, evaluate their compatibility with Enterprise Engineering, form a method of using smart contracts in the DEMO methodology and propose a software architecture of an IT system based on Enterprise Engineering integrating smart contracts.

Blockchain, decentralized, autonomous, secure and trustless database, with its smart contracts is a promising technology, with many proposed applications. Therefore, there exists misconceptions developed regarding its potential. In order to avoid this in our solution, we reviewed its capabilities and accounted for the common misconceptions. Consequently, we evaluated its compatibility with EE. In order to benefit fully from blockchain, two approaches were distinguished. Firstly, we stated that we can use blockchain as notarization system for information and documents related to transaction as well as transaction progress and results. Secondly, we used it as a transaction execution system for partial or full execution.

Secondly, we have shown how we can apply smart contracts to EE, a method of creating smart contract from DEMO models was devised. The method defines how to translate the models into contract code using the EE principles, for both approaches, notarization and execution. Then we proposed a software architecture of an Enterprise Information System integrating blockchain and basic principles of their interaction.

In the end a proof of concept was developed based on mortgage process. We have defined and analyzed the process, applied the DEMO methodology and consequently developed an Ethereum Solidity contract using our method, proving that our concept is feasible.

To summarize, we were able to devise a method of using smart contracts in the DEMO methodology in the context of EIS. Such integration would bring blockchain‘s benefits to process execution, namely secured and trustless storage of data and immutable transaction execution. This might bring a new way of looking at transactions with external actors, where the blockchain can

serve as a trustless coordination of the operation and notarized data source. The formalization of contract creation from DEMO models can also support their integrity and security, as the EVM in its turing-completeness or other smart contract implementations are a powerful tool, but also error-prone, as security incidents have been reported [27]. Further, it could serve as a methodology for creating well designed Dapps and DAOs.

## Future research

As for future work, deeper formalization of the method of translating the models into contract code could be considered as well as automatic creation of contracts by BPMS based on EE such as DEMO engine. Secondly, possibilities of optimization of the contract code should be examined to reduce the costs and effectiveness of blockchain execution.

Another topic to enhance is assets representation on blockchain. Consider how would our proof of concept mortgage process change, if the property and its ownership was represented digitally on blockchain instead of cadaster. That way, further atomization could be introduced and the importance of smart contract would gain new dimensions. DEMO methodology could also be used for such assets digitalization and creation of their blockchain representations.

In general, the proposed concept of using smart contracts would have many benefits, but there are some pre-requisites, such as digital identity or payments in cryptocurrencies, that would need to be fulfilled for their broader application. As the technology is still at its beginning, it is important to monitor its future advance.

# Bibliography


[1] Dietz, J. L. MI-MEP DEMO Bachelor [lecture]. *EDUX ČVUT FIT*, 2016, [online], [cit. 2017-03-26].

[2] Vejrážková, Z. *Business Process Modeling and Simulation: DEMO, BORM and BPMN*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technologies, 2013.

[3] Dietz, J. L. *Enterprise Ontology: Theory and Methodology.* Berlin Heidelberg New York: Springer, 2006, ISBN 3-540-29169-5.

[4] bitcoin.org. Bitcoin Developer Guide [online]. [cit. 2017-05-1]. Available from: `https://bitcoin.org/en/developer-guide#block-chain`

[5] Smart Contracts Alliance. Smart Contracts: 12 Use Cases for Business & Beyond. 2016, [online], [cit. 2017-10-08]. Available from: `http://www.the-blockchain.com/docs/Smart%20Contracts%20-%2012%20Use%20Cases%20for%20Business%20and%20Beyond%20-%20Chamber%20of%20Digital%20Commerce.pdf`

[6] Weber, I.; Xu, X.; et al. Untrusted Business Process Monitoring and Execution Using Blockchain. In *Business Process Management: 14th International Conference, BPM 2016*, Cham: Springer International Publishing, 2016, pp. 329–347.

[7] Skotnica, M. *Towards the Foundations of Fact and Rules Ontology for Discrete Systems*. Master's thesis, Czech Technical University in Prague, Faculty of Information Technologies, 2016.

[8] Hintzen, J.; van Kervel, S. J. H.; et al. A professional Case Management System in production, modeled and implemented using DEMO. In *Proceedings of 16th IEEE Conference on Business Informatics (CBI)*, 2014.

[9] Swan, M. *Blockchain*. Sebastopol: O'Reilly Media, Inc, 2015, ISBN 978-1-491-92049-7.

[10] Dietz., J. L. *Demo Basis - Glossary of Terms*. Sapio bv, 2014.

[11] Dietz, J. L. The PSI theory - understanding human collaboration. 2016, [online], [cit. 2017-04-18].

[12] Antonopoulos, A. M. *Mastering Bitcoin*. Sebastopol: O'Reilly Media, Inc, 2014, ISBN 978-1-449-37404-4.

[13] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. [online], [cit. 2017-04-28]. Available from: `https://bitcoin.org/bitcoin.pdf`

[14] NEO. NEO Smart Contract Introduction. [online], [cit. 2018-1-2]. Available from: `http://docs.neo.org/en-us/sc/introduction.html`

[15] Swanson, T. Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. 2016, [online]. Available from: `http://www.ofnumbers.com/wp-content/uploads/2015/04/Permissioned-distributed-ledgers.pdf`

[16] Pilkington, M. Blockchain Technology: Principles and Applications. In *Research Handbook on Digital Transformations*, edited by F. X. Olleros; M. Zhegu, 2015. Available from: `https://ssrn.com/abstract=2662660`

[17] Ethereum. Ethereum Project. [online], [cit. 2017-11-29]. Available from: `https://ethereum.org/`

[18] The Linux Foundation. Hyperledger. [online]. Available from: `https://www.hyperledger.org/`

[19] Ubiq Technologies Inc. Ubiq. Available from: `https://ubiqsmart.com/`

[20] NEO. NEO White Paper. Available from: `http://docs.neo.org/en-us/`

[21] Dannen, C. *Introducing Ethereum and Solidity*. Brooklyn, New York, USA: O'Reilly Media, Inc, 2017, ISBN 978-1-4842-2534-9.

[22] Ethereum. Ethereum Homestead Documentation. [online], [online], [cit. 2017-11-92]. Available from: `http://www.ethdocs.org/en/latest/index.html`

[23] Rikken, O. 3 Smart Contract Misconceptions. [online], [cit. 2017-12-03]. Available from: `https://www.coindesk.com/3-common-smart-contract-misconceptions-explored/`

[24] Bartoletti, M.; Pompianu, L. An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, Ap ril 7, 2017, Revised Selected Papers*, Cham: Springer International Publishing, 2017, ISBN 978-3-319-70278-0, pp. 494–509.

[25] ethernodes.org. The Ethereum Nodes Explorer. [online], [cit. 2017-12-03]. Available from: `https://www.ethernodes.org`

[26] Clack, C. D.; Bakshi, V. A.; et al. Smart Contract Templates: foundations, design landscape and research directions. *CoRR*, volume abs/1608.00771, 2016, `1608.00771`.

[27] Norta, A. Designing a Smart-Contract Application Layer for Transacting Decentralized Autonomous Organizations. In *Advances in Computing and Data Sciences: First International Conference, ICACDS 2016, Ghaziabad, India, November 11-12, 2016, Revised Selected Papers*, Singapore: Springer Singapore, 2017, ISBN 978-981-10-5427-3, pp. 595–604.

[28] Weske, M. *Business Process Management, 2nd Edition*. Berlin, Heidelberg: Springer„ 2012, ISBN 978-3-642-28615-5.

[29] White, S. A. Introduction to BPMN. [online], [cit. 2017-12-11]. Available from: `http://www.bptrends.com/publicationfiles/07-04%20WP%20Intro%20to%20BPMN%20-%20White.pdf`

[30] García-Bañuelos, L.; Ponomarev, A.; et al. Optimized execution of business processes on blockchain. In *International Conference on Business Process Management*, Springer, 2017, ISBN 978-3-319-64999-3, pp. 130–146.

[31] Ethereum Solidity. Solidity Documentation. [online], [cit. 2017-11-22]. Available from: `https://solidity.readthedocs.io/en/develop/index.html`

[32] Drwiega, T. JSON RPC. [online], [cit. 2017-11-22]. Available from: `https://github.com/ethereum/wiki/wiki/JSON-RPC`

[33] Remix. Remix - Solidity IDE. [online], [cit. 2017-11-22]. Available from: `https://remix.readthedocs.io/en/latest/`

[34] MGIC for Homebuyers. The mortgage process. [online], [cit. 2017-11-21]. Available from: `https://homebuyers.mgic.com/getting-your-mortgage/mortgage-process.html`

[35] Hands on Banking. Steps in the lending process. [online], [cit. 2017-11-21]. Available from: https://handsonbanking.org/adults/buying-home/getting-mortgage/steps-in-the-lending-process/

[36] Ethereum Solidity. Common Patterns. [online], [cit. 2017-11-27]. Available from: http://solidity.readthedocs.io/en/develop/common-patterns.html

Appendix A

# Acronyms

**ABD** Actor Bank Diagram

**AM** Action model

**ATD** Actor Transaction Diagram

**BCT** Bank Contents Table

**BPD** Business Process Diagram

**BPM** Business Process Management

**BPMN** Business Process Model and Notation

**BPMS** Business Process Management Systems

**CM** Construction model

**DACs** Decentralized autonomous corporations

**DAOs** Decentralized autonomous organizations

**Dapps** Decentralized applications

**DEMO** Design and Engineering Methodology for Organizations

**EE** Enterprise Engineering

**EIS** Enterprise information system

**EOA** Externally Owned Account

**EVM** Ethereum Virtual Machine

**FM** Fact model

**FTP** File Transfer Protocol

**HTTP** Hypertext Transfer Protocol

**IAM** Interaction model

**ISM** Interstriction model

**IT** Information Technology

**IUT** Information Use Table

**MDE** Model Driven Engineering

**OCD** Organization Construction Diagram

**OER** Organization Essence Revealing

**OFD** Object Fact Diagram

**OPL** Object Property List

**PM** Process model

**PSD** Process Structure Diagram

**SMTP** Simple Mail Transfer Protocol

**SC** Smart Contract

**TPT** Transaction Product Table

Appendix B

# Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `solidity` ..... solidity implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
- `models` ..... DEMO models images