TÉCNICO LISBOA

Implementation of DEMO Action Model in Blockchain Smart Contracts

Marta Maria Simões Aparício

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Pedro Manuel Moreira Vaz Antunes de Sousa
Prof. Sérgio Luís Proença Duarte Guerreiro

Examination Committee

Chairperson: Prof. Francisco António Chaves Saraiva de Melo
Supervisor: Prof. Pedro Manuel Moreira Vaz Antunes de Sousa
Member of the Committee: Prof. André Ferreira Ferrão Couto e Vasconcelos

January 2021

Acknowledgments

My first words of thanks could not fail to go to my advisors, Professor Pedro Sousa, and Professor Sérgio Guerreiro. They presented me with an interesting and challenging topic that I had never worked on before. Thanks for everything, especially the insight, support, guidance, and sharing of knowledge that made this dissertation possible. I hope we keep in touch.

I would also like to thank project QualiChain and Fundação para a Ciência e Técnologia (FCT) for their financial support in the publication of research, carried out within the scope of this dissertation. Project QualiChain supported by the European Commission program H2020 under the grant agreement 822404, and FCT supported by national funds with reference UIDB/50021/2020 (INESC-ID).

I also want to thank Marek Skotnica. Despite our physical distance, we managed to help each other. I liked being able to ”travel” in pandemic times, even without having done so. Our meetings went beyond research topics, which allowed me to exchange ideas as well as meet a friend. I am glad we have eternalized our mutual interest in politics in a paper. I hope to see you when you come to Portugal, or I go to the Czech Republic, whichever comes first.

I am sure that I would forget someone if I were to list the names of the friends I met at the Instituto Superior Técnico, not because there are many, but because I have a bad memory. I know that I made a friend on the first day that I entered here for the first time, and even in my master's degree, I managed to make friends with people with whom I shared three years of bachelor's degree, but with whom I had never interacted. From the first to the last, you all helped me in stressful moments. But above all of you made me laugh a lot, and I keep those laughs as the best memories I take from here.

Now it's time to thank my extended family, my longtime friends. All your patience and understanding in times of greatest stress were fundamental. In times of pandemic, all your messages of encouragement in ”caps lock” and full of emojis motivated me. Thanks for everything.

I would also like to thank my parents for their infinite patience and faith in me. The values that you pass me on are foundations of my personality that I cherish and hope to keep forever. I will forever be grateful for your efforts and dedication to our family.

Lastly, but not least, I want to thank my sister, who always leads the way. I know that because you are the oldest, you will never understand the profound admiration I have for you. Thank you for always

making me grow and through you being able to see the future that lies ahead.

A meritorious thanks also to Lúcifer ”Tito” Bonifácio for the serenity he brings to my life.

To each and every one of you – Thank you.

“*There is no demand for women engineers, as such, as there are for women doctors; but there's always a demand for anyone who can do a good piece of work.*” - Edith Clarke

Abstract

Blockchain offers a decentralized ledger that records transactions and allows tracking assets in a secure and reliable way. While Blockchain can guarantee that the stored data cannot be tampered with, it cannot guarantee that the data was correct when it was stored in the chain. Rectifying incorrect information is almost impossible in the Blockchain. What is needed is a correctness check for data before data is stored in the Blockchain. Enterprise Engineering is a scientific discipline with an underlying methodology of modeling processes, DEMO. The DEMO methodology offers all the information about the process needed to evaluate and create Smart Contracts. By computing a high-level formal model as DEMO Action Model and generating Smart Contracts from it, one can create Smart Contracts that ensure the correctness of transactions data before it is stored in the Blockchain. This solution allows the reuse of Ontological models and guarantees the correct implementation of Smart Contracts. This work, whose methodology is based on DSRM, relates concepts between DEMO and Solidity and creates a base Smart Contract that encapsulates the DEMO concepts. Regarding the mapping between the DEMO concepts and Solidity concepts, DEMO ensures the syntactic validation, while the two Use Case ensure the semantic validation. As for the base contract, it instantiates the mapping, and its validation is done using the EVM compiler and Solidity language. The work was communicated to ICEIS, KEOD, and MODELSWARD to obtain a scientific evaluation.

Keywords

Blockchain; Smart Contract; DEMO; DEMO Action Model; Ethereum; Solidity

Resumo

A Blockchain oferece um ledger descentralizado que regista transações e permite o rastreamento de ativos de maneira segura e confiável. Enquanto a Blockchain pode garantir que os dados armazenados não podem ser adulterados, ela não pode garantir que os dados estavam corretos quando foram armazenados. O que torna necessário uma verificação de exatidão dos dados antes de estes serem armazenados na Blockchain. Enterprise Engineering é uma disciplina científica com uma metodologia subjacente de modelagem de processos, DEMO. A metodologia DEMO oferece todas as informações sobre o processo necessário para avaliar e criar Smart Contracts. Ao elaborar um modelo formal como o DEMO Action Model e gerar Smart Contracts a partir dele, pode-se criar Smart Contracts que garantem a exatidão dos dados de transações antes de serem armazenados na Blockchain. Esta solução permite a reutilização de modelos ontológicos e garante a correta implementação de Smart Contracts. Este trabalho, seguiu a metodologia DSRM, relaciona conceitos entre DEMO e Solidity e cria um Smart Contract base que encapsula os conceitos DEMO. Em relação ao mapeamento entre os conceitos DEMO e os conceitos de Solidity, o DEMO garante a validação sintática, enquanto os Use Cases garantem a validação semântica. Já o contrato base instância o mapeamento, e a sua validação é feita utilizando o compilador EVM e a linguagem Solidity. O trabalho foi comunicado a ICEIS, KEOD e MODELSWARD para obtenção de uma avaliação científica.

Palavras Chave

Blockchain; Smart Contract; DEMO; DEMO Action Model; Ethereum; Solidity

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Acronyms

ARS	Action Rule Specifications
BP	Business Process
BPM	Business Process Management
BPMN	Business Process Model and Notation
BPMS	Business Process Management System
CRM	Customer Relationship Management
DEMO	Design & Engineering Methodology for Organizations
DMS	Document Management System
DS	Design Science
DSRM	Design Science Research Methodology
EBNF	Extended Backus–Naur Form
ebXML	Electronic business eXtensible Markup Language
EDI	Electronic Data Interchange
EE	Enterprise Engineering
EIS	Enterprise Information System
ERP	Enterprise Resource Planning
EO	Enterprise Ontology
EOA	Externally Owned Account
EVM	Ethereum Virtual Machine
FCT	Fundação para a Ciência e Técnologia
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol

ISO	International Organization for Standardization
IT	Information Technologies
J2EE	Java 2 Enterprise Edition
MDE	Model-Driven Engineering
SMTP	Simple Mail Transfer Protocol
WIS	Work Instruction Specifications
XML	Extensible Markup Language

1

Introduction

Contents

As enterprises are complex systems involving both human and technological aspects and are highly influenced by the environment in which they operate, architecting an enterprise and the information systems that support their functioning is not an easy task [7].

Each enterprise has to manage several processes. Business Processes (BPs) are what enterprises do whenever they deliver a service or a product to customers. Dumas et al. [8] defines a BP as "*a collection of inter-related events, activities, and decision points that involve a number of actors and objects, which collectively lead to an outcome that is of value to at least one customer*". Armed with this definition of a BP, Business Process Management (BPM), was also defined in [8], as a "*body of methods, techniques, and tools to identify, discover, analyze, redesign, execute, and monitor business processes to optimize their performance*".

Information Technologies (IT) play a significant role in BPs or processes in general, as more and more of them are supported by IT systems. Business Process Management Systems (BPMSs) are just one type of IT tools that supports the implementation and execution of BPs. There are many others, including Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) systems, and Document Management Systems (DMSs).

BPMSs are mainly focused on the automation of BPs within a given organization (intra-organizational processes). Many processes, however, span across organizational boundaries (inter-organizational processes). For inter-organizational processes, business partners need security mechanisms to ensure that the messages they exchange are authentic and confidential.

Blockchain technology can be defined as a decentralized ledger that records transactions and allows the tracking of assets securely and reliably [9]. A blockchain is decentralized, in the sense that it is composed of a set of nodes managed by different entities, so it is not controlled by any entity individually. Blockchain properties are particularly useful in inter-organizational processes where actors do not necessarily trust each other. Blockchain allows for an authentic and confidential exchange between BP parties through the use of digital signatures and encryption, respectively. Both mechanisms are based on public-key cryptography, where each partner has both a public key and a private key. Blockchain 2.0 introduces concepts for flexible and programmable transactions referred to as smart contracts.

1.1 Motivation

1.1.1 The need for contracts

Over the past two centuries, trade has grown significantly, completely changing the global economy. The integration of the national economy into the global economic system has been one of the most important developments of the last century [10]. This integration process often referred to as globalization, has been achieved in the significant growth of trade between countries. Trade transactions include both

goods and services. The supply chains of these goods and services are becoming increasingly complex and global, requiring some type of legal commitment between involved parties.

Since ancient times, binding agreements have been drawn up to recognize and manage the rights and obligations between parties. These binding agreements, for value exchange, are now called contracts and protect both parties. These contracts are written or spoken and require the parties to trust each other to fulfill their commitments.

1.1.2 The need for trustless technology

The continuous development of digitization and internationalization in various fields such as e-commerce or supply chain management [11] has led to an increasing shift to more collaboration in BPs. A characteristic of collaborative BPs is that they are composed of different sub-processes executed by various organizations. By their nature, sub-processes carried out by one organization are usually beyond the domain of influence of all other collaborators [12]. But in the context of the process, the results of the sub-process may be important to other collaborators. Therefore, from the perspective of all other collaborators, this creates uncertainty in the process. Where there is uncertainty, trust is needed [5]. Therefore, collaborative BPs are trust-intensive in nature.

The intricate environment created by global trading leads to enforcing trust by centralized organizations such as insurances and banks, which themselves are supported by the ultimate centralized authority in the system, the state institutions. Nonetheless, this type of organization forces business partners to depend upon third parties to manage and enforce those contractual agreements. Adding to this, as stated in Section 1.1.1, this problem is particularly acute when there is a lack of trust between parties, and as known contracts require parties to trust each other to fulfill their commitments.

Blockchain technology can be seen as a solution to coordinate inter-organizational processes involving untrusted parties [8]. This technology provides a way to record something that happened to ensure that it cannot be deleted once recorded. Smart Contracts are mechanisms, that exist in latter Blockchain generations, that ensure that a given routine is executed every time a transaction, of a given type, is recorded. Traditionally, the coordination of all parties in a transaction is accomplished through message exchange. Instead of exchanging messages, all parties involved in the inter-organizational process can execute transactions on the Blockchain. This alternative method ensures that important business rules are always followed.

Through a distributed ledger and shared business logic in the form of Smart Contracts, enterprises can execute shared business logic and monitor the state of process instances [12]. Blockchain allows organizations to use a single shared database to manage and operate their shared BP.

From a more historical perspective, in 2009, Satoshi Nakamoto proposed a peer-to-peer network where transactions are hashed into an ongoing chain, forming a record that cannot be changed without

redoing the Proof-of-Work [9]. Later on, a Blockchain called Ethereum[1] was specifically created and designed to support Smart Contracts. The Smart Contract concept was first introduced in 1997 by Nick Szabo [13]. Nowadays, Smart Contracts are enforced by the network of computers that makes up the Blockchain, guaranteeing the adequate execution of the code that composes the Smart Contracts, based on proven cryptographic algorithms.

Opportunities for applying Blockchain technology for BPM are discussed by Mendling et al. in [14]. An open-source BPMS that runs entirely on the Blockchain has been released, called Caterpillar [15]. This work, and others that will be discussed later on in Chapter 3, prove the feasibility of executing BPs on top of Blockchains by taking process models as a starting point [16].

Although the feasibility of implementing BPs on top of Blockchain has been proven, Blockchain is considered a relatively new technology compared to others. As Gartner claims [17], Blockchain technology is still very immature to support most of the potential use cases and there's still a tremendous amount of research, implementation, and adoption to be done. Further, building systems on Blockchain is non-trivial due to the steep learning curve of the Blockchain technology. According to another survey by Gartner [18], "*23 percent of [relevant surveyed] CIOs said that blockchain requires the most new skills to implement of any technology area, while 18 percent said that blockchain skills are the most difficult to find*" [18].

A concrete and real example of a problem that occurred in a Blockchain was analyzed and discussed by Alex Norta in [19]. The paper refers to a crowdfunding project that was hacked due to security flaws, which resulted in a loss of $50 million. Concerning this problem, North states "*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 smart-contract management. Instead, we propose in this keynote paper that is crucial to address a gap for secure smart-contract management pertaining to the currently ignored application-layer development*" [19]. This suggests that the automatic generation of Smart Contracts would have the added value of creating a new level of security.

Concerning the automatic generation of Smart Contracts, several approaches can be followed, one of which being a Model-Driven Engineering (MDE) approach. MDE is a software engineering method that uses models with various views and levels of abstraction to achieve different goals in the software development process. Models with a lower level of abstraction can be used to directly generate software production code.

In the context of Blockchain applications, MDE is of particular importance as the Blockchain is by design an immutable record, so it is non-trivial or even infeasible to update Smart Contracts [20].

There are already tools that resort to MDE to facilitate the development of Blockchain applications in the space of BPs and Blockchain applications. An example of that is Lorikeet [21] an MDE tool that can

[1] https://ethereum.org/

automatically generate Smart Contracts from BP models and/or registry data schema. Chapter 3 will further analyze this tool and other tools relevant to the topic.

1.1.3 The need for enterprise ontology

In today's world, is difficult to think of an enterprise that hasn't yet transitioned from analog to digital (Digitization); improved BPs by leveraging digital technologies (Digitalization); or even leveraged emerging technologies to build new business systems, business models and others (Digital Transformation).

Indeed, the fast-evolving world puts high demands on enterprises in terms of the constant need for flexibility, adaptation, and innovation to create new business opportunities [22]. The more enterprises rely on IT support for their functioning, the more the same flexibility, adaptation, and innovation are required from enterprise information systems. As stated in Postulate 1 of the Enterprise Engineering Manifesto [2]: "*In order to perform optimally and to implement changes successfully, enterprises must operate as a unified and integrated whole*".

Therefore, IT should be an integral part of the enterprise strategy. The field of Enterprise Engineering (EE) takes care of the integral design of an enterprise. One of the main goals of EE is to offer a global perspective as well as full alignment between IT and businesses.

According to research by Hoogervorst [23], many studies have shown that the proportion of successful implementation of strategic plans is less than 10%. Hoogervorst also cited many studies that show that lack of coherence and integration are the most important reasons for such a high failure rate. This indicates that coordination between enterprise elements is mandatory in order to develop a solid enterprise strategy.

Dietz uses the ψ-theory [24] (underlies the notion of Enterprise Ontology (EO)) to construct a methodology providing an ontological model of an organization, i.e. a model that is coherent, comprehensive, consistent, and concise, and that only shows the essence of the operation of an organization model [25].

This methodology is called Design & Engineering Methodology for Organizations (DEMO). DEMO has been widely accepted in both scientific research and practical appliance [3, 26]. In DEMO, an enterprise is seen as a system of people and their relations, authority and responsibility. The usage of a strongly simplified models that focus on people forms the basis of DEMO. By using a language that is common in the enterprise, the understanding of such models are guaranteed, even though they're abstract and have a conceptual nature.

The EO model reduces complexity [27]. This reduction in complexity makes the organization more transparent. It also shows the consistency between all areas within the enterprise, such as BPs and workflows.

[2] http://www.ciaonetwork.org/publications/EEManifesto.pdf

1.2 Problem Definition

The intricate environment created by global trading leads to enforcing trust by centralized organizations such as insurances and banks which themselves, are supported by the ultimate centralized authority in the system, the state institutions. Enterprises are left to no other option than relying on these institutions to manage transactions. A potential problem is that these institutions constitute a single point of trust and failure. This means they constitute a single place to attack in case of malicious intent.

Instead of having a single point of trust and failure, Blockchain offers a decentralized ledger that records transactions and allows tracking assets in a secure and reliable way. A blockchain is decentralized, in the sense that it is composed of a set of nodes managed by different entities, so it is not controlled by any entity individually. Each block contains a list of transactions, and the sequence of blocks is replicated in all nodes. Blockchain and Smart Contracts are touted as a solution to better trust and verify assets. Using Blockchain technology requires some knowledge, which can prevent the adoption of this technology in enterprises.

While Blockchain can guarantee that the stored data cannot be tampered with, it cannot guarantee that the data was correct when it was stored in the chain. Rectifying incorrect information is extremely complex, if not impossible, in a blockchain, and the absence of a central authority makes it difficult to prevent fraudulent entries or hold the fraudster accountable. What is needed is a correctness check for data before data is stored in the blockchain.

EE is a scientific discipline with an underlying methodology of modeling processes, DEMO. DEMO has the theoretical foundation of an enterprise and a methodology on how to capture its essence and processes.

The DEMO methodology fulfills the C_4E quality criteria and, in its complexity, offers all the information about the process needed to evaluate and create smart contracts. From the *Action Model*, it is possible to infer transaction states, their order, and names. Furthermore, it's possible to know what information is needed, in each transaction, what are the relations between them, their flow, and action rules.

By computing a high-level formal model as DEMO *Action Model* and generating Smart Contracts from it, one can create smart contracts that ensure the correctness of transactions data before it is stored in the blockchain. This solution allows enterprises to experience the possibilities of blockchain without needing to know all sorts of technical details. Ultimately, this solution allows the reuse of Ontological models and guarantees the correct implementation of Smart Contracts.

1.3 Dissertation Goals

This dissertation intends to infer if by using Ontology, namely, DEMO Action Models, it is possible to extract automatically the knowledge required to generate Smart Contracts on Blockchain. If the hypothesis

ends up being admissible, through demonstration, verification and validation, this will mean the reuse of Ontological models in line with the correct implementation of the Smart Contract on Blockchain. Given the motivation presented, the research questions addressed by this dissertation are:

- How can a Blockchain Smart Contract be used to implement a DEMO Action model?
- What will be the ontological implications of implementing DEMO Action Models through Blockchain Smart Contracts?

This dissertation sets out to address the hypothesis of using Blockchain Smart Contracts to implement DEMO Action Models. DEMO methodology presents a system to capture the essence and model processes. Blockchain, on the other hand, presents a new technology that can be used for the implementation of processes. The underlying intention is to determine the possible cooperation between them. Furthermore, the formalization of principles for the translation of DEMO Action Models into Solidity Smart Contract is the greatest contribution this dissertation intends to reach.

Generating a Blockchain Smart Contract from this Ontological Model saves money, time, and lowers the risk of error, and serves as a useful tool for non-programmers who need to implement a Blockchain Smart Contract. This work helps make the specification of contracts between parties more reliable and reachable by any party who so desires. This contribution will mainly affect non-technical people, that do not comprehend software code, and without this knowledge are once again dependent on a centralized organization to write the contracts.

Modeling methods based on Ontology can lead to better data standards, business practices, and processes for developing and operating a Blockchain. The modeling methods based on Ontology can aid in the formal specifications for automated inference and verification in the operation of a Blockchain. This last benefit is particularly interesting as is very similar to the definition of Smart Contracts as "*pieces of software that represent a business arrangement and execute themselves automatically under predetermined circumstances*" [28]. For these reasons Kim and Laskowski claimed in [29] that "*ontology-based blockchain modeling will result in a blockchain with enhanced interpretability*" [29].

1.4 Research Methodology

The research questions introduced in Section 1.3 will be answered using Design Science Research Methodology (DSRM) [1]. The DSRM incorporates principles, practices, and procedures required to conduct researches of Information Systems. The Design Science (DS) process includes six steps: problem identification and motivation (Section 1.2 and Section 1.1), the definition of the objectives for a solution (Section 1.3), design and development (Chapter 4), demonstration (Section 5.2 and Section 6.2), evaluation (Section 5.4 and Section 6.4), and communication (Chapter 7), Figure 1.1. DS is of

importance to create artifacts successfully.

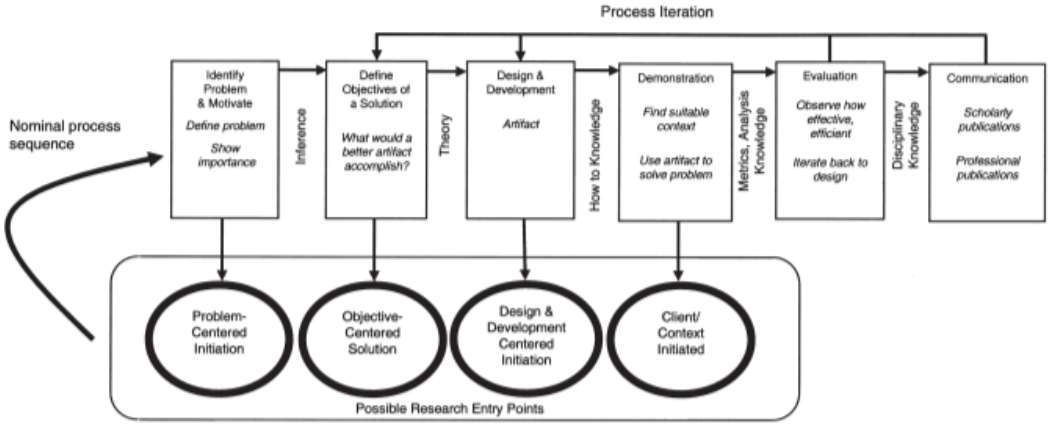

Figure 1.1: Design Science Research Methodology Process Model. Image retrieved from [1].

This is of great importance as, although an activity-centric approach is suitable for describing the activities carried out by the process agent and the relationship between them, this method has great disadvantages [30]. In an activity-centric BP model, the link between the control flow and the data required for it or produced by it is not clear [12]. In 2003, IBM recognized this issue and developed a process modeling approach centered around business artifacts [31]. The artifact-centric approach has been recognized as a potentially superior method to manage inter-organizational processes [30, 32].

1.5 Dissertation Structure

This document is structured as follows. First, the document starts with an introduction to the thesis and adopted research methodology. The identified problem and the motivation for this work are also presented in Chapter 1. Chapter 3 contains the state-of-the-art relatively to the problem of this research. Chapter 4 explains in detail the assumptions and mapping that constitutes the proposed solution. This chapter is followed by a demonstration of the mapping which is instantiated in Ethereum Blockchain, prototypes in the context of a Rent-A-Car Use Case in Chapter 5 and the context of the European Union Parliament Elections Use Case in Chapter 6. The last two chapters also evaluate the proposed mapping. Chapter 7 contains the communication of the works developed to international conferences to obtain a scientific evaluation. Finally, in Chapter 8, the conclusion of this work and topics that need to be solved in future work are stated.

2

Background

Contents

This chapter explains the key background concepts relatively to BPMS, Blockchain, Smart Contracts, MDE,EO, DEMO and DEMO *Action Models*.

2.1 Business Process Management Systems Historical Background

BPMSs are mainly focused on the automation of BPs within a given organization (intra-organizational processes). For inter-organizational processes, business partners need security mechanisms to ensure that the messages they exchange are authentic and confidential. Once these security mechanisms are in place, business partners will have to agree on a common format for the messages to be exchanged. Business partners can rely on Electronic Data Interchange (EDI) standards, which specify the format documents exchanged between enterprises. Finally, the last level of inter-organizational integration consists in defining the choreography of message exchanges between partners. A choreography defines the sequence of message exchanges that are to take place between a set of business partners.

Due to the widespread of contracts, to mediate business interactions, standardization has been made. The International Organization for Standardization (ISO)[1] is an international standard-setting body composed of representatives from various national standards organizations. ISO 15000 suggests a standardization to Electronic business eXtensible Markup Language (ebXML)[2]. ebXML is designed to create a global electronic market place where enterprises of any size, anywhere can find each other electronically and conduct business. ebXML is preferred over EDI, as by using the economies of scale presented by the internet, breaks through EDIs costs and implementation difficulties [2].

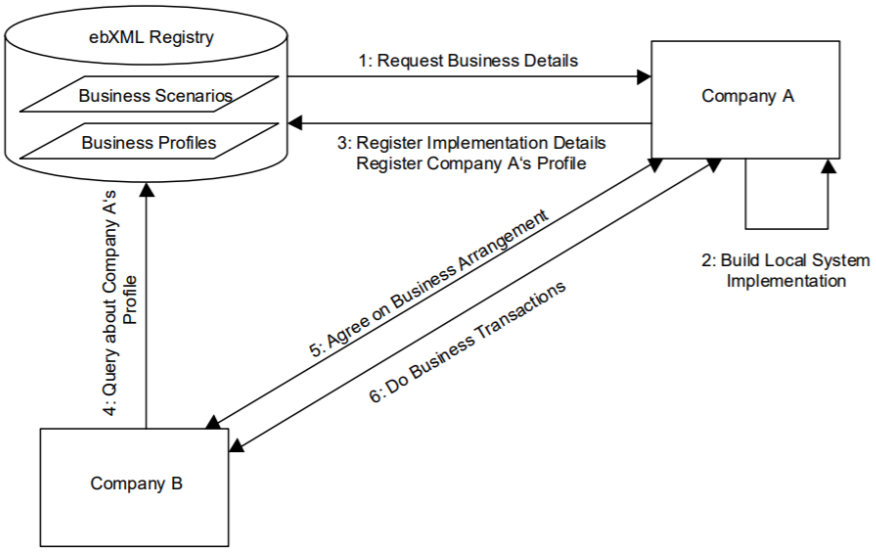

Figure 2.1: An Overview of ebXML System Architecture. Image retrieved from [2].

[1] https://www.iso.org/
[2] http://www.ebxml.org/

Figure 2.1 shows a diagram on how organizations prepare for ebXML, search for new trading partners, and then engage in electronic business. Company A requests business details from the ebXML registry (step 1) and decides to build its own ebXML-compliant application. Company A submits its business profile information to the ebXML registry. The business profile submitted to the ebXML registry describes the company's ebXML capabilities and constraints, as well as its supported business scenarios. Company B, which uses an ebXML-compliant shrinkwrapped application, discovers the business scenarios supported by Company A in the registry (step 4). Company B sends a request to Company A stating that they would like to engage in a business scenario (step 5). Before engaging in the scenario, company B submits a proposed business arrangement directly to Company A's ebXML-compliant software interface. The proposed business arrangement outlines the mutually agreed upon business scenarios and specific agreements. Company A then accepts the business agreement. Company A and B are now ready to engage in e-business using ebXML (step 6).

2.2 Blockchain

Most people know Bitcoin[3], making it a buzzword, but few understand it. What is meant by this is that few understand the underlying technologies and paradigms behind Bitcoin – namely Blockchain, and its wide variety of applications [33, 34]. Blockchain can be seen as a paradigm shift or just a hype [35], either the case, it made people understand that transactions no longer belong exclusively to the non-digital world.

As the name suggests, Blockchain is a chain of blocks that contains information. This technique was originally described in 1991 by a group of researchers and was originally intended to timestamp digital documents so that it was impossible to tamper them [36]. It went by mostly unused until it was adapted by Satoshi Nakamoto, in 2009, to create the digital cryptocurrency Bitcoin [9]. Satoshi Nakamoto described Blockchain as an architecture that gives participants the ability to perform electronic transactions without relying on trust.

It started as the usual framework of coins made from digital signatures, which allows strong control of ownership. But still required a trusted third party to prevent the Double-Spending Problem, which was solved by a peer-to-peer network approach. This network timestamps transactions by hashing them into an ongoing chain of hash-based Proof-of-Work [37], forming a record that cannot be changed without redoing the Proof-of-Work. What makes this possible is that each block contains some data, the hash of the block, and the hash of the previous block, as shown in figure Figure 2.2.

The data that a block stores inside depends on the type of Blockchain. Blocks normally store the details of multiple transactions, each with an identification for the sender, the receiver, and the asset. A block also has a hash that identifies its content, and this hash is always unique. If something changes

[3] https://bitcoin.org/en/

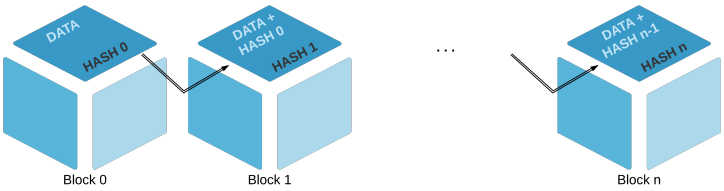

Figure 2.2: Representation of the sequence of elements that make up a Blockchain.

inside a block, that will cause the hash to change. These are the reasons why hashes are very useful to detect changes in blocks. The hash of the previous block effectively creates a chain of blocks, and this is the technique that makes a Blockchain so secure. Each block also contains the chain's Merkle root or "*hash of all hashes*" which prevents the need to download the entire chain to verify the validity of the chain for each transaction. This last element was not represented in figure Figure 2.2, since it is not relevant for understanding the Blockchain concept. However, the hashing technique is not enough, with the high computational capacity that exists today, where a computer can calculate hundreds of thousands of hashes per second. To mitigate this problem, Blockchains has a consensus mechanism called Proof-of-Work. This mechanism slows down the creation of new blocks since if the temper of a block occurs, the Proof-of-Work of all the previous blocks has to be recalculated.

So, the security of Blockchain comes from its creative use of hashing and a Proof-of-Work mechanism. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll outpace attackers. Of course, its distributive nature also adds a level of security, since instead of using a central entity to manage the chain, Blockchain uses a peer-to-peer network where anyone can join. Information is broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest Proof-of-Work chain as proof of what happened while they were gone.

Blockchains can operate in three forms: *public*, *private* or *hybrid*. A *public* Blockchain, like Bitcoin, is an open Blockchain whereby anyone can participate by reading or sending transactions or by joining the consensus process through an anonymous node. *Public* Blockchains offer maximum transparency, and its main goal is to prevent the concentration of power. In contrast, some industries only transact with trusted peers. To fit those industries, *private* Blockchains have emerged. *Private* Blockchains that operate in a more closed environment, making the Blockchain seemingly more attractive to entities that do not like the idea that users are anonymous and that the consensus process is performed by anonymous nodes. Participants prefer that write permissions are privileged to a single organization, while read permissions may be public or restricted to an arbitrary extent. *Hybrid* Blockchains utilize a consensus process that is controlled by a pre-selected set of nodes to validate transactions instead of a strict *public*/*private* dichotomy.

2.3 Smart Contract

The term "*Smart Contract*" was first used by Nick Szabo (Computer science, law scholar and cryptographer) in 1997 [13], long before the creation of Bitcoin. The primitive idea was to use a distributed ledger to store contracts. Nick Szabo drew up the term "*Smart Contracts*" and explained to them as follows: "*I call these new contracts "smart" because they are far more functional than their inanimate paper-based ancestors. No use of artificial intelligence is implied. A smart contract is a set of promises, specified in digital form, including protocols within which the parties perform on these promises*" [13].

In the context of Blockchain, in particular second-generation Blockchains, Smart Contracts are just like contracts in the real world. The only difference is that they are completely digital, in the sense that they are both defined by software code and executed or enforced by the code itself automatically without discretion.

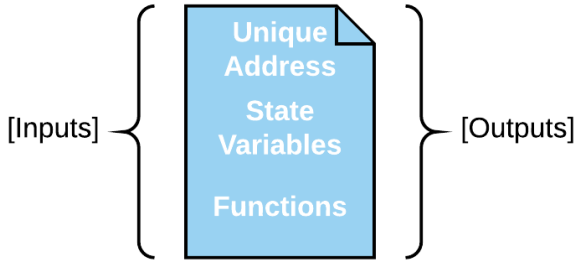

Figure 2.3: Representation of the elements that compose a Smart Contract.

The trust issue is also addressed once Smart Contracts are stored on a Blockchain, they inherit some interesting properties: immutable and distributed. Being immutable means that once a Smart Contract is created, it can never be changed again. So, it isn't possible to tamper with the code of the contract. Being distributed means that the output of the contract is validated by every node on the network. So, a single node cannot force the behavior of the contract since it is dependent on the other nodes.

Like all algorithms, Smart Contracts may require input values and only act if certain predefined conditions are met. When a particular value is reached the Smart Contract changes its state and executes the functions, that are programmatically predefined algorithms, automatically triggering an event on the Blockchain. If false data is inputted to the system, then false results will be outputted [38]. For a better understanding of the elements that compose a Smart Contract see figure Figure 2.3.

Blockchains cannot access data outside of their network, thus require some form of trusted data feed as input to the system called an Oracle [39–41]. An Oracle is a data feed provided by an external service and designed for use in Smart Contracts on the Blockchain. Oracle's provide external data and trigger Smart Contract executions when predefined conditions are met, such conditions could be any data. An Oracle in the context of Blockchains and Smart Contracts is then an agent that finds and verifies real-

world occurrences and provides this information to a Blockchain to be used by Smart Contracts. Oracle's are third-party services that are not part of the Blockchain consensus mechanism thus, the source of information needs to be trustworthy.

Smart Contracts that are nothing more than flexible and programmable transactions that run on top of Blockchain enable the creation of more complex Decentralized Applications (DApps), and even Decentralized Autonomous Organizations (DAOs) [20, 40, 41]. The term DApps refers to open-source autonomous applications that execute across decentralized network nodes [42]. When multiple DApps are interconnected in an autonomous decentralized manner, a DAO may be created. DAOs 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*" [43].

2.4 Model-Driven Engineering

Although the feasibility of implementing BPs on top of Blockchain has been proven, Blockchain is considered a relatively new technology compared to others [17, 18].

While Blockchain can guarantee that the stored data can't be tampered with, it can't guarantee that the data was correct when it was stored in the chain. Rectifying incorrect information is extremely complex, if not impossible, in a blockchain. What is needed is a correctness check for data before data is stored in the blockchain. By computing a high level formal model as DEMO Action Model and generating Smart Contracts from it, one can create smart contracts that ensure the correctness of transactions data, before it is stored in the blockchain. This suggests that the automatic generation of Smart Contracts would have the added value of creating a new level of security required.

Concerning the automatic generation of Smart Contracts, several approaches can be followed, one of which being a MDE approach. MDE is a software engineering method that uses models with various views and levels of abstraction to achieve different goals in the software development process. Models with a lower level of abstraction can be used to directly generate software production code. In the context of Blockchain applications, MDE is of particular importance as the Blockchain is by design an immutable record, so it is non-trivial or even infeasible to update Smart Contracts [20]. MDE against other approaches presents the following gains:

- MDE tools can generate well-tested code implementing best practices and help developers manage software complexity by only focusing on building high-level models without requiring expert development knowledge [44], thereby reducing the occurrence of vulnerable code that may easily

be attacked.

- With research on Blockchain technology on the rise and new ways of applying this technology to different domains being discovered [45], it becomes necessary not to over-fit to a specific Blockchain platform. Model abstraction can avoid this problem, and Model-Driven development tools can be applied at multiple Blockchain platforms.

- Models are easier to understand than code, which improves development efficiency [4]. It is easier to check the correctness of a model, and Model-Driven tools can ensure that the deployed code is not modified after it is generated from the model.

- MDE development of Blockchain applications can facilitate communication with domain experts [21], who can examine models to understand how their ideas are represented in the system.

2.5 Enterprise Ontology

EE is an emerging discipline that takes an engineering perspective on enterprises. It aims to address the problems in organizations that cannot be solved by the traditional organizational sciences. The conceptual pillars are EO, Enterprise Design, and Enterprise Governance. It serves to provide a basis for the common understanding of some area of interest among a community of people who may not know each other at all and who may have very different cultural backgrounds [25].

EO presents four distinction axioms: the operation, transaction, composition, and distinction axioms. The *operation axiom* states that an enterprise is composed of subjects who may perform two kinds of acts, production and coordination acts, which result in production and coordination facts. Whereas the performance of an act is a direct or indirect contribution to the bringing about of a product or service, the creation of facts are state transitions that cannot be undone. A state transition generates an event at a certain point in time. In a BP, organizational agents perform coordination acts, which result in coordination facts in the form of commitments.

Building upon the *operation axiom*, the *transaction axiom* defines standardized patterns in which production and coordination acts and facts occur. These patterns are called transactions. A transaction consists of three phases, namely the order, execution, and result phase. A transaction may be canceled at certain points in time. This cancellation can occur due to the actors have a different claim to truth, which must be proved by facts, or due to norms of the actors. The transaction axiom is highly relevant for inter-organizational BPs, as every process may be mapped to a set of transactions.

The third axiom states that every transaction is either a part of another transaction or is self-activated. This axiom is called the *composition axiom*. According to this axiom, a BP is a collection of causally

related transactions. A process may then either be externally activated by a customer or self-activated by an internal actor.

The *distinction axiom* states that humans have three distinct abilities with regards to coordination and production acts. They are the *performa*, *informa*, and *forma* abilities. Transactions typically occur on one or more of these levels. On the *performa* level, *ontological* transactions establish new things, such as the creation of a product or making a decision. On the *informa* level, *infological* transactions serve to realize the *ontological* transaction, in which exchanges of information or knowledge between actors happen. On the *forma* level, *datalogical* transactions take place to realize *infological* exchange by storing or transmitting data.

In order to really cope with the current and the future challenges, of the social and economic structure, conceptual models of the enterprises are needed. These models are known as ontological models, and most comply with these five properties: *coherence*, *comprehensiveness*, *consistency*, *conciseness*, and *essence*, collectively abbreviated as C_4E.

By *coherence* it means that the distinguished aspect models constitute a logical and truly integral whole. By *comprehensive* it means that all relevant issues are covered, that the whole is complete. By *consistent* it means that the aspect models are free from contradictions or irregularities. By *concise* it means that no superfluous matters are contained in it, that the whole is compact and succinct. The most important property, however, is that this conceptual model is *essential*, that it shows only the essence of the enterprise. In particular, it means that the model abstracts from all realization and implementation issues [46].

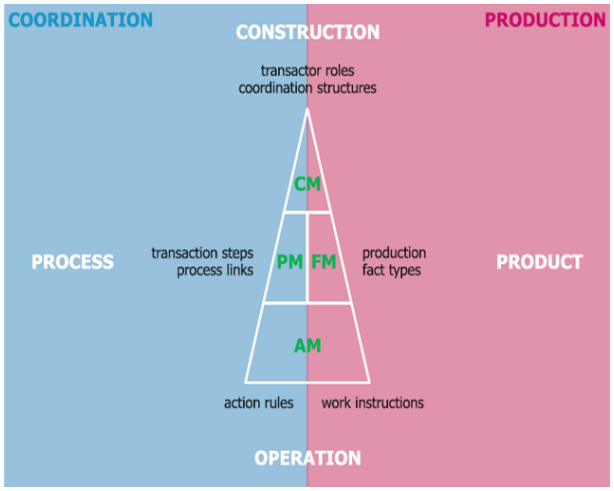

Figure 2.4: The integrated DEMO aspect models. Image retrieved from [3].

A complete enterprise ontology consists of four related aspect models: the *Cooperation Model*, the *Process Model*, the *Action Model*, and the *Fact Model*, as presented in Figure 2.4.

The *Cooperation Model* is the most concise model and specifies the identified transaction types and

the associated actor roles (with specific authority and responsibility, but not a specific person), as well as the information links. The *Process Model* contains, for every transaction type in the *Cooperation Model*, the specific transaction pattern of the transaction type. It also contains the causal and conditional relationships between transactions. The *Action Model* specifies the action rules that serve as guidelines for the actors in dealing with their agenda. The *Fact Model* specifies the state space of both the production world and the coordination world of the enterprise.

An ontology may be seen as a branch of philosophy that is concerned with the nature and structure of existence. Aristotle even defines ontology, in [47], *"as the science of "being qua being," i.e., the study of attributes that belong to things because of their very nature"* [48].

With ontology, it's possible to make a common interpretation of data across organizations, making it easier and cheaper to exchange transactions. This interpretation may be accomplished via informal data standards, for instance, conventions or business practices. Or via more formal specifications that enable automated reasoning and verification through software applications that execute between enterprises. Ontological modeling allows capturing entities and their relations in the real world, allowing to capture many different domains [49].

2.6 Design & Engineering Methodology for Organizations (DEMO)

DEMO is an enterprise modeling methodology for (re)designing and (re)engineering organizations. In DEMO, an enterprise is a system of people and their relations, authority, and responsibilities. The usage of strongly simplified models that focus on people forms the basis of DEMO. By using a language that is common in the enterprise, the understanding of such models is guaranteed, even though they're abstract and have a conceptual nature. DEMO describes the construction and operation of the organization while completely abstracting from implementation and realization. The core concepts of DEMO are a transaction and the ψ-theory.

Dietz and Mulder [3] introduced the concept of EO as a collection of capabilities to deal with the essential aspects of a process-based organization. DEMO aims to understand the organization using models that are essential for the design and operational of the organization. Both the DEMO and EO concept were largely introduced in Section 1.1.3.

DEMO describes a pattern for the communication and production of acts and facts that occur between actors in the scope of a business transaction. According to ψ-theory, in the standard pattern of a business transaction exists two actor roles, the initiator and the executor (Figure 2.5).

The transaction pattern is performed by a sequence of coordination and production acts that leads to the production of the new production fact. The standard pattern of a transaction comprises three phases: *(i)* the Order phase that contains the acts of request, promise, decline and quit; *(ii)* the Execution phase

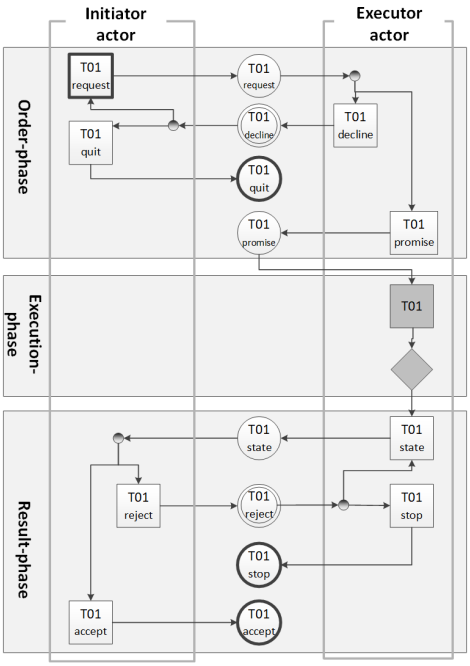

Figure 2.5: DEMO standard pattern transaction between two actors with separation between communication and production acts. Image retrieved from [3].

that contains the act execution, that includes the production act of the new fact itself; and *(iii)* the Result phase that contains the acts declare, reject, stop and accept.

The following four steps are present in every transaction and represent the happy flow: request, promise, state, accept. Each of these steps can be revoked anytime during the transaction. The other party can either allow this, or refuse it.

2.7 Action Model

The Action Model of an organization is the ontological model of its operation. For every internal actor role, it contains the rules that guide the role fillers in doing their work. The guidelines for responding to coordination facts (C-facts) are called action rules; the guidelines for performing production acts (P-acts) are called work instructions. They are respectively expressed in Action Rule Specifications (ARS) and Work Instruction Specifications (WIS). Because work instructions are usually enterprise-specific, they will not be elaborated [3].

The specification of an action rule is divided into three sequential parts: event part, assess part, and response part. The event part comprises a when-clause, optionally supplemented by a with-clause and a while-clause. In the when-clause, one states the type of event that is going to be settled. The optional while-clause specifies the event that impedes settling the event in the when-clause. Then the rule cannot

be executed until both events have occurred. Thus, there may be several action rules for the same kind of event, but each of them with a specific impending event kind. There cannot be two or more action rules that have the same when-clause and while-clause. Otherwise said, actors never have to choose what rule to execute, there will at most be one [6]. The assess part consists of several propositions whose truth value must be determined. The propositions are divided in three: rightness, sincerity, and truth. In the rightness division, the conditions are specified that must apply to the participating actors, and that serve to ensure that they have the proper authority. The claim to sincerity is specific for the individual fillers of the actor roles, it seems hard to make the sincerity division more precise than checking if they seem to be trustworthy. Lastly, the propositions in the truth division always regard the state of the production world. They serve to check the possible violation of existence and occurrence rules in the production world [6]. The response part starts with the most distinctive condition in DEMO action rules, compared to other business rules approaches: "*'if performing the action after then is considered justifiable'*". This sentence expresses the fundamental autonomy of the executor to decide what the best response is, given the specific case he/she is dealing with [6].

In Algorithm 2.1, the ARS are defined in Extended Backus–Naur Form (EBNF).

Algorithm 2.1: Action Rule Specifications defined in EBNF. Retrieved from [6].

begin

action rule specification = event part, assess part, response part;

event part= agendum clause, [while clause], [with clause];

agendum clause = "**when**", transaction kind name, "**for**", entity variable, "**is**", perfect tense intention;

while clause = "**while**", transaction kind name, "**is**", perfect tense intention−;

with clause = "**with**", {(property variable, indefinite entity reference) | (attribute variable, indefinite value reference)}−;

assess part = rightness division, sincerity division, truth division;

rightness division = "rightness:" {property condition | attribute condition}− | <informal specification>;

sincerity division = "sincerity:" {property condition | attribute condition}− | <informal specification>;

truth division = "truth:" {property condition | attribute condition}− | <informal specification>;

<informal specification>= a text between "*" and "*";

response part = "**if**", "performing the action after **then** is considered justifiable", "**then**", action clause, ["**else**" action clause];

action clause = {present tense intention, transaction reference, {with clause}}−;

transaction reference = transaction kind name, "**for**", entity type name;

3

Related Work

Contents

This chapter includes part of the definition of the objectives for a solution step of DSRM, in which the state-of-art, the problem, and solutions are presented to infer goals of the solution proposed Chapter 4 to the problem defined Chapter 1.

3.1 Implications of Cooperation between Ontology and Blockchain

Mohanta et al.'s work [33] defines some of the issues of the technologies used in this work. The issues are scalability, flexibility, and privacy. Regarding the Smart Contracts, privacy is considering the fact that the scripts are available publicly in the network which it may not be suitable in some application. However, this may not be seen as an issue anymore, since as mentioned in section 2.2, Blockchains can operate in three forms: public, private, or hybrid. This paper also identifies seven different use cases of smart contracts and Blockchain-based applications (Supply Chain, Internet of Things, Healthcare System, Digital Right Management, Insurance, Financial System, and real estate), the same later on encountered on [50].

Lead by Mohanta et al. [33], the work of Kim et al. [29] was read since it believed that ontologies could contribute to developing Blockchain applications. Kim et al. suggest that ontology-based Blockchain modeling would result in a Blockchain with enhanced interpretability, stating even that "*modeling approach based on formal ontologies can aid in the formal specifications for automated inference and verification in the operation of a blockchain*" [29]. Kim et al.'s work only reinforced the initial idea of this work that a modeling approach based on formal ontologies can aid in the development of Smart Contracts that execute on the Blockchain. Going further, Kim et al. even did a Proof-of-Concept, where the translation of TOVE Traceability Ontology axioms into Smart Contracts that could execute a provenance trace and enforce traceability constraints on the Blockchain.

To better understand the usability of these concepts and how to implement them in a real-world context, the "*A Pattern Collection for Blockchain-based Applications*" [51] was read. This paper had a set of patterns for the design of Blockchain-based applications. The pattern that most relates to this work was the "*Patern 3: Legal and Smart Contract Pair*", where a bidirectional binding is established between a legal agreement and the corresponding Smart Contract. The idea behind is for the Smart Contract to digitalize the conditions defined on the agreement. Thus, these conditions can be checked and enforced automatically by the Smart Contract. However, not all legal terms can be easily digitalized. Smart Contracts can also enable automated regulatory compliance checking in terms of the required information and process. Nonetheless, the capability of compliance checking might be limited due to the constraints of Smart Contract programming languages.

The last limitation stated in regards to the constraints of Smart Contract programming languages guides the way to encounter the work of Idelberger et al. [52]. Their work suggests that combinations

of logic-frameworks and Blockchain systems may lead to Smart Contracts that are easier to work with for technical and non-technical people. These combinations may also lead to new opportunities for applications for these logic-frameworks. At last, this work states that logic and procedural approaches are not incompatible. These approaches have the potential to complement each other advantageously. By providing a declarative specification of the content of the contract, to be complemented with a procedural definition of the steps necessary, to fulfill the obligations of the contract.

"*Towards a Blockchain Ontology*" [53] and "*Understanding the Blockchain Using Enterprise Ontology*" [54], both from Joost de Kruijff and Hans Weigand, used Enterprise Ontology and DEMO to describe the Blockchain Ontology from a Datalogical, Infological, and Essential (Business) perspective. This paper suggests that by specifying the Blockchain application on the business level first, it will be possible to generate the Blockchain implementation automatically, with some design parameters to be set.

In particular, [54] highlights the distinction axiom as highly relevant for the Blockchain. Following the three abilities already mentioned in section Section 2.5, three ontological layers are distinguished. The Datalogical layer describes Blockchain transactions at the technical level in terms of blocks and code. From there, the Infological abstraction is used, to describe the Blockchain transactions as effectuating an (immutable) open ledger system. This layer aims to abstract from the various implementations that exist today or will be developed in the future. To describe the economic meaning of the Infological transactions the Essential layer is used. This last is the preferred level of specification for a Blockchain application as it abstracts from the implementation choices.

The subject matter in this work are Smart Contracts, so it should be noted that these are seen as a type of transaction at the Datalogical abstraction layer. In regards to the Infological level, the Smart Contracts are enforced by rules of engagement that are implemented as Blockchain code. At last, for the Essential level, a contract is an agreement between agents consisting of mutual commitments. A Smart Contract is a contract in which commitment fulfillment is completely or partially performed automatically. Kruijff et al. didn't validate their Ontology, and the focal point was the Blockchain as a whole, unlike our work that focuses on Smart Contacts. Validation is pivotal to test and improve the capability of the proposed Ontology, to reduce Blockchain's ambiguity and inconsistency.

3.2 From Code-Centric to Model-Centric Development

As already mentioned, Smart Contracts supported by Blockchain technology are a somewhat novel field, so research on the topic is not very extensive. However, there have been attempts at raising the level of abstraction from code-centric to model-centric Smart Contracts development. The different approaches tried so far can be divided into three: the Agent-Based Approach, the State Machine Approach, and the

Process-Based Approach [55].

The Agent-Based Approach described by Frantz and Nowostawski [56] propose a modeling approach that supports the semi-automated translation of human-readable contract representations into computational equivalents to enable the codification of laws into verifiable and enforceable computational structures that reside within a public Blockchain. They identify Smart Contract components that correspond to real-world institutions and propose a mapping using a domain-specific language to support the contract modeling process. The concept Grammar of Institutions [57] is used to decompose institutions into rule-based statements. These statements are then compiled in a structured formalization. The statements are constructed from five components, abbreviated to ADICO:

- **A**ttributes – describes an actor's characteristics or attributes.
- **D**eontic – describes the nature of the statement as an obligation, permission or prohibition.
- A**I**m – describes the action or outcome that this statement regulates.
- **C**onditions – describe the contextual conditions under which this statement holds.
- **O**r else – describes consequences associated with non-conformance.

Using these components statements the execution of the smart contract is made. The statements are then linked by the structure of Nested ADICO [58], a variant of ADICO in which the institutional functions are linked, by the operators AND, OR, and XOR to create a simple set of prescriptions. The set of prescriptions is then transformed into a contract skeleton which has to be finished manually. Furthermore, it is argued that the Grammar of Institutions invites non-technical people to the Smart Contract development process.

The State Machine Approach bases on the observation that Smart Contracts act as state machines. A Smart Contract is in an initial state, and a transaction transitions the contract from one state to the next. The possibility of Smart Contracts as state machines is also described in the Solidity specification [1]. Mavridou and Laszka [59] show that the transformation of the Finite State Machine to Solidity [2] is partly automated since to ensure Solidity code quality, some manual coding might be necessary or added through plugins.

Through an article, "*Blockchains for Business Process Management - Challenges and Opportunities*" [60], were found Process-Based Approaches. Both DEMO and Business Process Model and Notation (BPMN) [3] are well established for modeling business processes. Weber et al. [16], describes a proposal to support inter-organizational processes through Blockchain technology. Captured in BPMN, large parts of the control flow and business logic of inter-organizational business processes can be compiled

[1] https://docs.soliditylang.org/en/v0.4.24/common-patterns.html#state-machine
[2] https://solidity.readthedocs.io/en/latest/
[3] http://www.bpmn.org/

from process models into Smart Contracts that ensure that the joint process is correctly executed. So-called trigger components allow the connection of these inter-organizational process implementations to Web services and internal process implementations. These triggers serve as a bridge between the Blockchain and enterprise applications. Weber et al. developed a technique to integrate Blockchain into the choreography of processes to maintain trust. The Blockchain enabled to store the status of process execution across all involved participants, as well as to coordinate the collaborative business process execution. Validation was made against the ability to distinguish between conforming and non-conforming traces. Besides, they also compared latency for public and private Blockchains, concluding that public Blockchains present higher latency.

Garcia-Banuelos et al. [61] introduces an optimization for Weber et al. [16]. In this work, to compile BPMN models into a Smart Contract in Solidity Language, the BPMN model is first translated into a reduced Petri Net. Only after this first step, the reduced Petri compiles into a Solidity Smart Contract. Compared to Weber et al. [16] this work [61] managed to decrease the amount paid of resources and achieve higher throughput. Another similar optimization was proposed by Wen Hu et al. [62], where first, the BPMN business process model is extended to the Petri net. It is then simplified according to the characteristics of Petri, the combination of nodes in the BPMN model that can be, regarded as the fusion task are found, and these fusion nodes are simplified. This method can write the optimized Smart Contract template while ensuring the integrity of the original business process.

Caterpillar, first presented in [15] and further discussed in [41], is an open-source BPMS that runs on top of the Ethereum Blockchain. Like any BPMS, Caterpillar supports the creation of instances of a process model (captured in BPMN) and allows users to track the state of process instances and to execute tasks. The specificity of Caterpillar is that the state of each process instance is maintained on the Ethereum Blockchain, and the workflow routing is performed by Smart Contracts generated by a BPMN-to-Solidity compiler. Caterpillar implements a comprehensive mapping from BPMN to Solidity. Given a BPMN model (in standard Extensible Markup Language (XML) format), it generates a Smart Contract (in Solidity), which encapsulates the workflow routing logic of the process model. Specifically, the Smart Contract contains variables to encode the state of a process instance and scripts to update this state whenever a task completes or an event occurs. Caterpillar supports not only basic BPMN control flow elements (i.e. tasks and gateways) but also includes advanced ones, such as sub-processes, multi-instances, and event handling.

Lorikeet [21], on other hand, can automatically create well-tested Smart Contract code from specifications that are, encoded in the business process and data registry models based on the implemented model transformations. The BPMN translator can automatically generate Smart Contracts in Solidity from BPMN models while the registry generator creates Solidity Smart Contract based on the registry models. The BPMN translator takes an existing BPMN business process model as input and outputs

a Smart Contract. This output includes the information to call registry functions and to instantiate and execute the process model. The registry generator takes data structure information and registry type as fields, and basic and advanced operations as methods, from which it generates the registry Smart Contract. Users can then deploy the Smart Contracts on Blockchain. This work builds upon already seen works, such as [16, 61], for the BPMN translation algorithms. The overall system architecture can be seen in Figure 3.1.

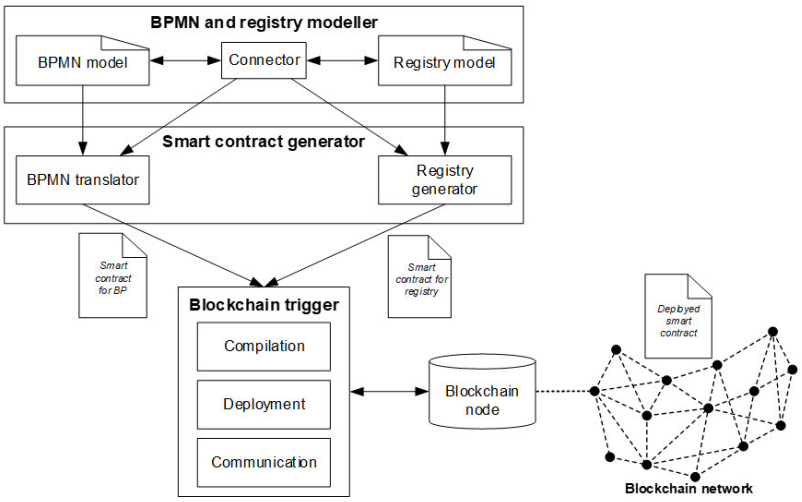

Figure 3.1: Architecture of Lorikeet. Image retrieved from [4].

The architecture of Lorikeet, presented in Figure 3.1, consists of a BPMN and registry modeler, a Smart Contract generator, and a Blockchain trigger.

However, BPMN doesn't adequately support the articulation of business rules that would be essential for the automatic generation of Smart Contracts. Process modelers normally, have to use workarounds, usually by also using additional tools, to include business rules in their processes [63]. Though frameworks such as the one presented on "*Converting DEMO PSI Transaction Pattern into BPMN: A Complete Method*" [64], which present an original method for converting DEMO process models into a BPMN 2.0 notation, would be possible make use of these works in order to fulfill or goals presented in Section 1.3. For a more straightforward comparison between approaches and respective tools please see Table 3.1.

Müller et al. [5] presents a taxonomy to classify the trust-enhancing capabilities of the blockchain technology in collaborative BPs. It goes further in understanding how the blockchain technology relates to trust to analyze how it can improve trust-intensive collaborative BPs. Then it discusses how blockchain technology can reduce uncertainties, reduce vulnerabilities, and build confidence in a process. Based on the established blockchain-based trust patterns, the impact of the technology as a trust-enhancing tool was analyzed. As summarized in Figure A.2, with the hash-based storage of data objects, data integrity can be verified at any point in time. A smart contract can facilitate the execution of an activity. The transparent event log ensures the integrity of events. The correct execution of business logic for

gateways, sequence, and message flows can be enforced, by a blockchain-based business process engine or similar working smart contracts. Blockchain can be seen as a highly available computing tool as a decentralized way to execute programming logic encoded in smart contracts in a peer-to-peer network. The transparent event log pattern ensures non-repudiation of executable process elements.

3.3 Cooperation-Oriented Research between Ontology and Blockchain

"*Using Blockchain to Improve Collaborative Business Process Management: Systematic Literature Review*" [65] presents a systematic review, which identifies and analyses the state-of-the-art research papers about collaborative BPM in Blockchain domain. This paper follows Kitchenham's method [66], which allows locating different types of proposals that address Collaborative BPs using Blockchain. These studies have been also classified according to the activities of the BPM lifecycle to which they offer support. For this purpose, Dumas' [8] BPM lifecycle has been used to perform this classification. However, it is possible to conclude that the current efforts by the scientific community in integrating Blockchain into BPM are at a very early stage. Despite this situation, this systematic review shows that the international scientific community has an important and growing interest to address the improvement of process management using Blockchain technology. This paper believes the use of a MDE paradigm could be interesting to apply to Blockchain technology to reduce costs and improve quality. In short, this paper presents a comprehensive review of blockchain technology and its applications in the domain of collaborative process management.

B-MERODE [67], uses a MDE approach to also generate smart contracts supporting cross-organizational BPs. Although the validation is done, by modeling a practical example, a rice supply chain, a large-scale application would be an essential next step to strengthen the validation and find some potential weaknesses. B-MERODE uses six distinct models as a consequence of the layered approach, which inevitably separates several concerns instead of representing them into a smaller set of models. B-MERODE is based, in MERODE [68] by belonging to an artifact-centric paradigm and provide good coverage of the BALSA dimensions [69]. MERODE uses a set of layers and a set of models and integrates all the different models in a consistent way providing a holistic specification of the system being developed. However, B-MERODE presents a tremendous benefit compared to other approaches. B-MERODE is entirely platform-independent B-MERODE can generate smart contracts that can be executed on any blockchain and can benefit from the existing expertise on MERODE transformations.

Although B-MERODE [67] seems to correspond to the objectives, this solution does not take into account the perspective of resources and organization, whereas the solution presented in Chapter 4 has [70]. From a resource perspective, understand descriptions of resources within the enterprise together with their capabilities [70]. And from an organizational perspective, understand structural relationships

of organizational units together with responsibilities and authorities [70].

Hornackova et al. [40] work proved to be relevant due to its alignment with the purpose of this work. Its *Further Research*, refers to one of the things this present work is trying to reach: formalization of principles of translating DEMO models into contract code. In "*Exploring a Role of Blockchain Smart Contracts in Enterprise Engineering*" [40], principles for creating Smart Contracts from all DEMO aspect models are described, and the software architecture of an IT system based on Enterprise Engineering integrating Smart Contracts is proposed. Also, a Proof-of-Concept implementation of a Smart Contract of a mortgage process using a DEMO methodology was developed, to demonstrate the feasibility of the proposed concepts. Proving this way that Blockchain Smart Contracts is usable in the implementation of an Enterprise Information System (EIS) based on DEMO methodology. Further, shares the same belief that by applying the DEMO methodology, a reduction of unwanted states, the prevention of errors, and an improvement of security, which is crucial for Smart Contracts since they are representing valuable assets, can be reached. However, Hornackova et al.'s work never formalizes the principles or even applies these beliefs to a sizable sample of use cases.

"*An ontological analysis of artifact-centric business processes managed by smart contracts*" [12], presents an ontological analysis of inter-organizational BP artifacts managed by blockchain-based smart contracts. This work presents a mapping between the three ontological layers. On the essential level, process participants conduct acts during standard interaction patterns called transactions. These acts result in facts, which are related to physical or artificial business objects. On the infological level, agents invoke services in process contracts, which maintains the state of artifact lifecycles and data objects. An act on the essential level maps to one or more service invocations on the infological level. In turn, the resulting essential fact maps to a state change on the infological level. On the infological level, agents invoke services that transform the state of business artifacts. State transitions are bound by business rules contained in the process contract. On the datalogical level, we see that smart contracts contain programmable functions and state variables. Smart contracts are deployed on a particular blockchain. The infological layer depicts how agents localize and transform information about business artifacts. A service invocation on the infological level maps to a transaction initiation on the datalogical level. Infological service invocations are bound by business rules contained in the process contract and are represented by datalogical functions. The function execution is validated before the successful addition of a transaction in a block.

	C. K. Frantz & M. Nowostawski	I. Weber & et al.	A. Mavridou & A. Laszka
Underlying Concept	Grammar of Institutions	Business processes	Smart contract as State Machine
Abstraction	ADICO Statements	Business Process Specification	Finite State Machine Model
Model Transformation	Semi-Automated	Semi-Automated	Automated
Direct Transformation	No	No	Yes
Transformation Result	Skeleton Contract	Smart Contract	Smart Contract
Advantages	Natural language can be interpreted by non-technical users; Formal syntax reduces ambiguity.	Business process specifications often already exist; Actors are distinguished; BPMN widely used.	Formal syntax; Make concepts from FSM and SC easily relatable; Usage of patterns against vulnerability.
Disadvantages	Complex way to structure natural language, almost a programming language; Not fully automated, the development stage is still manually executed.	Transformation from BPS to petri net to Solidity is complex; Aimed specifically at collaborative business processes, negating other usage.	Modeling complete and correct FSM's is a complex task; Roles are not explicitly stated; Some information is left implicitly.

Table 3.1: Comparison between Model-Driven approaches to generate Smart Contracts.

4

Proposal

Contents

The present chapter corresponds to the design and development phase of DSRM, in which is presented the DSRM artifacts. The artifacts aim at solving the identified problem (Chapter 1).

4.1 Proposal Vision and Approach

Organizations are complex systems that involve human and technological aspects and are influenced by the context in which they operate. Organizations have intra-organizational and inter-organizational BPs. In order to manage inter-organizational BPss, actors need security mechanisms that guarantee authenticity and confidentiality. Inter-organizational BPs create uncertainty between the parties involved. Stakeholders are dependent on third parties to manage and enforce contracts. Typically these third parties are banks or even state institutions, like courts, for example. These third parties are the only points of failure for possible attacks.

Blockchain technology can be seen as a solution to coordinate processes as it is a decentralized solution that does not allow for adulteration of information after its execution. In addition, doesn't force the existence of trust between those involved in the process and does not force dependence on third parties. Smart Contracts are mechanisms, existing since the second generation of Blockchains that ensure that business rules are always followed. They allow to execute shared business logic and monitor the state of the process instance. Although Blockchain technology seems like an optimal solution, it does not guarantee that the data is correct before being attached to the Blockchain.

Rectifying the data after its attachment to the Blockchain is practically impossible. Therefore, verifying the correctness of the data is essential. This type of obstacle can lead to the non-adoption of the technology even if this is a plausible solution.

Pragmatically, the proposed solution is intended not only to support inter-organizational BPs but also to ensure the correctness of the data attached to the Blockchain before its correction is irreversible. The correctness of data will be guaranteed by the implementation of ontological models since the ontology allows to create of a formal and explicit specification of a conceptualization shared between organizations.

The approach followed in this proposal is to allow the creation of Smart Contracts from an ontological model, namely DEMO Action Models. Furthermore, intends to create an abstraction that allows this to be a solution that can be adopted in any scope. The proposal is presented more graphically in figure 4.1.

In the 4.1 there are three connections. Link "A" corresponds to the generation of the smart contract from DEMO action models. This procedure is done by the "Executor" because he is the one who defines the conditions in which the service or product is provided.

Link "B" corresponds to the two possible interactions between the "Initiator" and the Blockchain. The

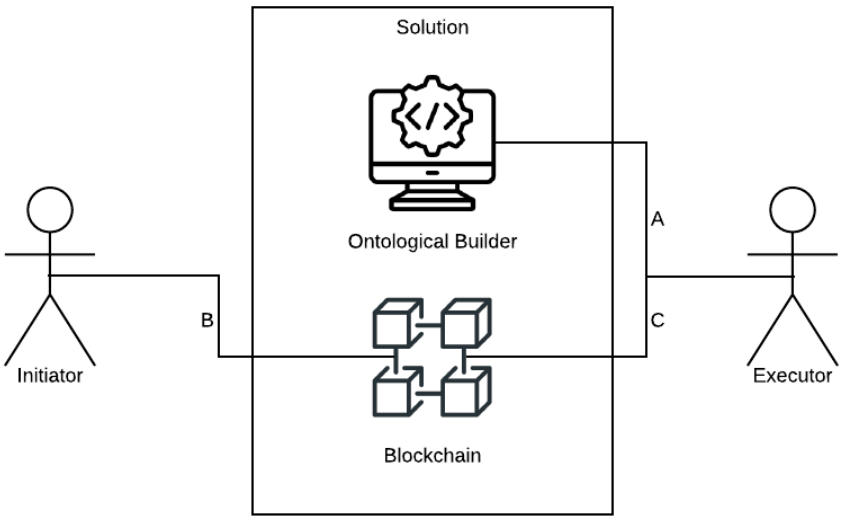

Figure 4.1: Proposal overview.

first interaction corresponds to the deployment of the Smart Contract, as this is the way to demonstrate the initial interest in interacting with the “Executor”. The second interaction corresponds to the invocation of the possible DEMO transaction steps to interact with the “Executor”.

Finally, link “C” corresponds to the invocation of DEMO transactions steps to interact with the “Initiator”, to the image of one of the interactions represented by link “B”.

4.2 Ontological Solution Hypothesis

There are many approaches to construct an ontology [71,72]. However, the main focus of this work is the choreography of activities within and between enterprises. Therefore, the adoption of EE was chosen. As part of EE, an EO is useful to create a formal and explicit specification of shared conceptualizations as further discussed in Section 2.5.

The concept of data independence designates the techniques that allow data to change without affecting the applications that process it [73]. It is the ability to modify a scheme definition at one level without affecting a scheme definition at a higher level. It is believed that a similar separation is highly needed for Smart Contracts to achieve the goals set in this work. DEMO is based on explicitly specified axioms characterized by a rigid modeling methodology and is focused on the construction and operation of a system rather than the functional behavior. It emphasizes the importance of choosing the most effective level of abstraction during information system development to establish a clear separation of concerns. The adoption of the *Distinction Axiom* of EO, presented in section Section 2.5, is proposed as an ontological basis for this separation.

At the *Essential* layer, an inter-organizational BP consists of several causally related transactions, as depicted by the *Composition Axiom* of EO. Each of these transactions has one initiator and one executor, which are roles played by process participants. Process participants perform acts during the various transaction phases. Whenever an act is carried out, it results in a fact. Whereas an act typically consists of a desirable future state, a fact states something true in the social world at a point in time [12]. In this layer, the present work believes a Smart Contract is a contract in which the commitment fulfillment is completely or partially performed automatically in Blockchain.

The acts on the *Essential* layer maps to one or more invocations to a contract on the *Infological* layer. This contract guards the invocation of business rules that can potentially change the state of that same contract. So, this work believes, for the *Infological* layer, a Smart Contract is enforced by a set of rules implemented on the Blockchain through code.

On the *Datalogical* layer, an inter-organizational process is invocations to smart contracts made by wallets. A transaction invocation, if successful, is recorded in a block. The execution of the function may emit an event and change state variables. So, for the *Datalogical* Layer, a Smart Contract is defined as a piece of code contained in nodes of the Blockchain.

4.3 Modeling Solution Hypothesis

One of the promising methods that can facilitate the development of Smart Contracts is MDE. Through MDE generates executable code for Smart Contracts from a set of models that specify the processes to be supported. For instance, [74] and [59] are tool-supported methods that allow the generation of Solidity Smart Contracts from BPMN diagrams and finite state machines, respectively.

Using MDE requires a set of models to be used as input for code generation. In particular, rather than focusing on algorithms and the elaboration of code, it focuses on the creation of models as representations of the software to be built. These models are then used to be transformed into other models or code. Key aspects of this transformational approach are that it speeds up the creation of software, that it enhances software quality, and that it facilitates the portability and interoperability of software. In *"Towards a shared ledger business collaboration language based on data-aware processes"* [75] called for the development of a shared ledger business collaboration language, which should be a language that business people can understand and use in cross-organizational BPs supported by blockchain technology.

One of the most common graphic process modeling notations used is BPMN. BPMN focuses on modeling the articulation between activities, resources, flows, gateways, events, messages, and data objects that occur in a BP.

However, and as already been stated in Section 3.2, BPMN doesn't adequately support the articula-

tion of business rules that would be essential for the automatic generation of Smart Contracts. Process modelers usually have to use workarounds, usually by also using additional tools, to include business rules in their processes [63]. In addition, Fickinger and Recker [76], concluded that BPMN has a level of 51.3% of overlapping language concepts and lacks state concept to ensure a more sound semantics.

DEMO, on the other hand, has been widely accepted in both scientific research and practical appliance [3, 26]. DEMO, sees an enterprise as a system of people and their relations, authority, and responsibility. DEMO ψ-theory specifies semantic meaning to the business transactions. The business transactions dynamic includes the actors, the communications, the productions, and all its dependencies. While BPMN does not describe any semantic for the business model, it only provides a set of constructs that could be combined accordingly with a specification.

4.4 Technology used in Solution Implementation

Blockchain technology introduces a decentralized, autonomous, replicated, and secure database. Based on cryptography offers a trustless network with no need for an intermediary, as presented in Section 1.1.2 and further explained in Section 2.2.

This work requires that the Blockchain chosen supports the creation of Smart Contracts. This requirement narrows the number of Blockchain platforms, as it makes it mandatory to be a second-generation Blockchain. By second-generation Blockchain platforms are meant Blockchains that offer the opportunity to build protocols, such as Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), and File Transfer Protocol (FTP), on top of the underlying TCP/IP transport layer of the Web.

There is a lot of different platforms that support the creation of smart contracts. Whether it is a public blockchain platform such as Ethereum or private such as Hyperledger [1]. Ethereum, with its Ethereum Virtual Machine (EVM) and Solidity programing language, is the most known platform bringing a standard that many other platforms, such as Hyperledger and Ubiq [2], build on. Moreover, Ethereum, due to its public nature, mimics better the real conditions of the prototypes presented on Chapter 5 and Chapter 6. As a public Blockchain often has definite limitations on transaction throughput per second. Which can lead to waiting situations for the submission of transactions to the ledger, hence decreasing the overall process's performance.

4.4.1 Ethereum

There are more than 800 Blochckain at the time of this writing. Ethereum is a platform for blockchain applications, with its blockchain and cryptocurrency, "*Ether*". It offers an environment to run decentralized

[1] https://www.hyperledger.org/
[2] https://ubiqsmart.com/

applications based on Smart Contracts. Ethereum does this by building what is essentially the ultimate abstract foundational layer: a blockchain with a built-in Turing-complete programming language, allowing anyone to write smart contracts where they can create their own arbitrary rules for ownership, transaction formats, and state transition functions. This abstract foundational layer is of great importance as already mentioned in Section 1.1.2, where the beliefs of Alex Norta in [19] are analyzed and discussed.

The keystone of Ethereum is the 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.*" [77]. 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 to maintain consensus of the blockchain.

In Ethereum, the state is made up of objects called "*accounts*" with each account having a 20-byte address and state transitions being direct transfers of value and information between accounts. An Ethereum account contains four fields: the nonce, a counter used to make sure each transaction can only be processed once; the account's current "*Ether*" balance; the account's contract code; and the account's storage. "*Ether*" is the main internal crypto-fuel of Ethereum and is used to pay transaction fees.

There are two types of accounts: Externally Owned Accounts (EOAs) and Contract Accounts.EOAs are controlled by private keys. An externally owned account has its own "*Ether*" balance, contains no code, and can transfer "*Ether*" or trigger contract code. These accounts are usually managed by wallets. On another hand, Contract Accounts are pieces of code in the EVM bytecode that live on the Ethereum Blockchain. They are usually written in higher-level languages like Solidity. Contracts have their own "*Ether*" balance, associated code, and persistent storage. Contracts can manipulate their storage, perform complex operations, and call other contracts. Contract Accounts perform operations that are activated by transactions from EOAs or messages from other contracts. Contract Accounts are accounts controlled by their contract code and are created when the contract is deployed on the Ethereum blockchain.

Transactions in Ethereum are signed data packages that store a message, sent by an EOAs to another account and are recorded on the blockchain. Messages can be sent between contracts and are performed locally on the EVM. Essentially, a message is like a transaction, except it is produced by a contract and not an external actor.

The Ethereum blockchain has a few test-nets. The Ropsten test-net [3] allows blockchain developments to test their work in a live setting, but without the need for real "*Ether*". This offers the ability

[3] https://ropsten.etherscan.io/

to consistently tweak the artifact being produced, without facing any serious gas fees or risk. Ropsten test-net is an exact copy of the real, main-net Network, and it allows anyone to engage and try without needing real "*Ether*" tokens. To experience all the features of Ropsten test-net and to launch or interact with it, there's a need for Ropsten "*Ether*", which is not real "*Ether*" but is still very useful as it is the fuel and currency of the Ropsten test-net.

4.4.2 Solidity

The Ethereum Blockchain was specifically created and designed to support Smart Contracts. Solidity is a high-level programming language to implement Smart Contracts specially design for the EVM. The smart contracts are deployed onto the blockchain in the form of a specialized bytecode. This bytecode then runs on each Ethereum node in EVM. Since creating smart contracts directly in the bytecode would be too impractical, multiple specialized high-level languages have been created, together with the compilers needed to convert them into EVM bytecode.

Solidity was chosen because it is developed under Ethereum, and it is the most used language for SCs for EVM. The building block in Solidity is a contract that is similar to a class in object-oriented programming. The contract contains persistent data in state variables, functions to operate on this data, and it also supports inheritance. The contract can further contain function modifiers, events, struct types, and other structures to allow the implementation of complex contracts and full usage of EVM and BC capabilities. A Smart Contract written in Solidity can be created either through an Ethereum transaction or by another already running contract, just like an instance of a class would be created. Either way, the contract code is then compiled to the EVM bytecode, a new transaction is created, holding the code and deployed to BC, returning the address of the contract for further interaction.

Solidity contracts can also be created, deployed, and interacted with programmatically using the JavaScript API web3.js [4], which is an Ethereum compatible library implementing the Generic JSON RPC specification [5], that provides a convenient interface for communication with Ethereum nodes.

4.4.3 Remix

Remix [6] is a browser-based IDE for creating smart contracts with an 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 [7] with a possibility to switch between virtual accounts and spend virtual "*Ether*" for full smart contract testing.

[4] https://web3js.readthedocs.io/en/v1.3.0/
[5] https://www.jsonrpc.org/specification
[6] https://remix-ide.readthedocs.io/en/latest/
[7] https://nodejs.org/api/vm.html

4.4.4 MetaMask

Metamask [8] is what makes it possible to switch between virtual accounts and spend virtual "*Ether*" for full smart contract testing. MetaMask is a browser plugin that serves as an Ethereum wallet, associated with an EOAs. It allows users to store "*Ether*", enabling them to make transactions to any Ethereum address.

4.5 Solution Architecture

This work starts with the belief a DEMO transaction is represented as a contract in a Blockchain. The contract has its address, internal storage, attributes, methods, and it is callable by either an external actor or another contract, as mentioned in section Section 2.3. This is the functionality needed to represent a DEMO transaction. Now, this present work argues that the DEMO *Action Model* could be implemented through Blockchain Smart Contracts without any support of the remaining DEMO models. It believes that the structure and content of a Smart Contract can be directly mapped to each action rule. By creating the action rules, the logic in which the smart contract operates is also being created.

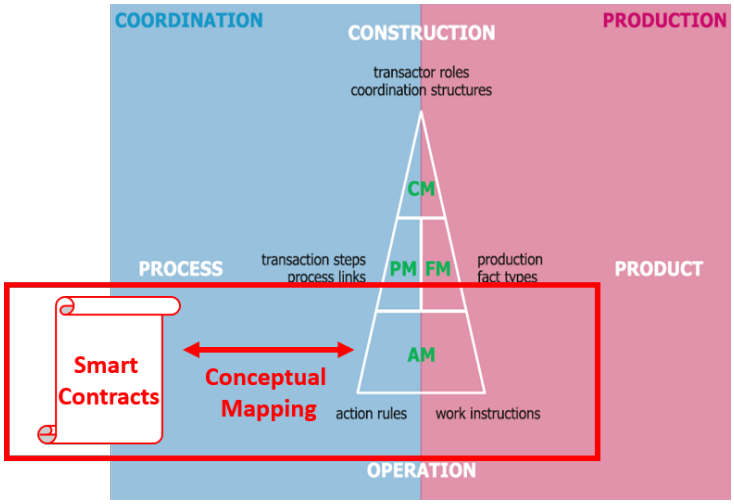

Figure 4.2: Solution Architecture.

The action rules contain all the decomposed detail of the above models, the basis of the DEMO methodology is exactly the *Action Model*, as can be seen in Figure Figure 4.2. The *Cooperation Model* specifies the construction of the organization, specifies the identified transaction types and the associated actor roles, as well as the information links between the actor roles and the information banks. By occupying the top of the triangle it is suggested that is the most concise model. The *Process Model* contains, for every transaction type in the *Cooperation Model*, the specific transaction pattern of the

[8] https://metamask.io/

transaction type. And, also contains the causal and conditional relationships between transactions. The *Process Model* is put just below the *Cooperation Model* in the triangle because it is the first level of detailing of the *Cooperation Model*, namely, the detailing of the identified transaction types. The *Action Model* specifies the action rules that serve as guidelines for the actors in dealing with their agenda. The *Action Model* is put just below the *Process Model* in the triangle because it is the second level of detailing of the *Cooperation Model*, namely, the detailing of the identified steps in the *Process Model* of the transaction types in the *Cooperation Model*. At the ontological level of abstraction, there is nothing below the *Action Model*. The *Fact Model* is put on top of the *Action Model* in figure Figure 4.2 because it is directly based on the *Action Model*; it specifies all object classes, fact types, and ontological coexistence rules that are contained in the *Action Model*.The *Action Model* is in a very literal sense the basis of the other aspect models since it contains all information that is (also) contained in the *Cooperation Model*, *Process Model*, and *Fact Model*; but in a different way. These models have as if a zoom in (*Action Model*) zoom out (*Cooperation Model*) relationship between each other. The *Action Model* is the most detailed and comprehensive aspect model.

4.6 Generate DEMO Action Model from Blockchain Smart Contracts

The implementation of the Action Rules of the *Action Model* is proposed using Smart Contracts. This is enabled by the ability of Smart Contracts within the scope of Blockchain technology to describe complex algorithms by using the Turing-complete programming language.

The proposed mapping not only takes advantage of the intrinsic properties of Blockchain technology but also takes advantage of some design patterns for Smart Contracts in the Ethereum Ecosystem [78].

Contracts often act as a state machine, which means that they have certain stages in which they behave differently or in which different functions can be invoked. A function call often ends a stage and transitions the contract into the next stage, this is known as the State Machine common pattern [78]. Through the DEMO theory, it is known that actors interact through creating and dealing with coordination facts (C-fact). Since these contracts will model interactions it seems fit to model C-facts into stages, these stages are implemented as Enums. Enums are a way to create a user-defined type in Solidity, for this particular application seven stages, corresponding to the C-facts: *Initial*; *Requested*; *Promised*; *Declined*; *Declared*; *Accepted*; *Rejected*. The *Initial* stage was created with the assumption that the deployment of a contract by someone doesn't mean they want to immediately start the transactions.

Function Modifiers can automatically check a condition before executing a function. So to guard against incorrect usage of the contract functions a dedicated function modifier will check if a certain function can be called in a certain stage.

Table 4.1: Correspondence between Solidity Smart Contract Components and DEMO Components.

DEMO Components	Solidity Components
C-fact	Enum
C-act	Function Invocation
P-fact	Event
P-act	Function Modifier

```
1  pragma solidity >=0.4.22 <0.7.0;
2  contract Transaction {
3      enum C_facts {Initial, Requested, Promissed, Declared, Accepted, Declined,
           Rejected }
4      C_facts public c_fact = C_facts.Initial;
5      address payable public initiator;
6      address payable public  executor;
7      event p_fact(address _from, bytes32 _hash);
8      modifier p_act() {
9          bytes32 hash = keccak256(abi.encodePacked(now, block.difficulty, msg.sender))
               ;
10         emit p_fact(msg.sender, hash);
11         _;           }
12     modifier atCFact(C_facts _c_fact) {
13         require(
14             c_fact == _c_fact,
15             "Function cannot be called at this time.");
16         _;                              }
17     modifier onlyBy(address _account) {
18         require(
19             msg.sender == _account,
20             "Sender not authorized.");
21         _;                              }
22     function nextCFact(bool happyFlow) internal {
23         if(happyFlow == true){
24             c_fact = C_facts(uint(c_fact) + 1);}
25         if(happyFlow == false && c_fact == C_facts.Requested){
26             c_fact = C_facts.Declined;}
27         if(happyFlow == false && c_fact == C_facts.Declared){
28             c_fact = C_facts.Rejected;}         }
29     modifier transitionNext(bool happyFlow) {
30             _;
31             nextCFact(happyFlow);           }
32 }
```

Listing 4.1: Base Contract Transaction.

The DEMO theory defines coordination acts (C-act) as acts in a business conversation, so these will be modeled as functions that can only be called by a certain address in a certain stage of the contract. To implement the guard of access to the functions the Restricting Access common pattern was implemented [78], through function modifiers once again. The production act (P-act) by the same logic is implemented through a function modifier that is only called in functions that represent the C-act *declare*. The function modifier that represents the P-act will emit an event corresponding to the production fact (P-fact), as a production fact is a result of performing a P-act.

To summarize, Table 4.1 shows the proposed correspondence.

Regarding the mapping presented in Table 4.1, the syntactic validation is guaranteed by DEMO, while the semantic validation is done by two Use Cases presented in Chapter 5 and Chapter 6.

Listing 4.1 represents a generic Transaction contract, from which all other transactions derive.

Since the building block in Solidity is a contract, that compares to a class in Object-Oriented programming, as referred in Section 4.4. In a more Object-Oriented manner, Listing 4.1 can be interpreted as the Base-Class from which the Sub-Classes inherit some of its functionalities.

The abstraction created allows for the contracts, seen as Sub-Classes, to re-use code from the Base-Class (Listing 4.1). Besides, the re-use of the functionalities allows for the Sub-Classes to only implement the business logic associated with it.

As discussed, the "*C-facts*" are represented through an Enum, with the seven stages considered as the coordination facts. The "*initiator*" and "*executor*" are represented through each of their addresses. The "*p-act()*" is represented by a function modifier that emits the "*p-fact*" event. At last, the "*atCFact()*" and "*onlyBy()*", are respectively modifiers of the common patterns State Machine and Restricting Access, already presented before. The "*transitionNext()*" modifier uses the "*nextCFact()*" function to control to switch between *C-facts*, checking if the execution followed an happy flow or not. The "_" symbol present inside the modifiers in Listing 4.1 is a place holder. The function body is inserted where the special symbol "_" in the definition of a modifier appears. This functionality allows executing modifiers after the correct execution of a function, as the "*transitionNext()*", or before, as "*onlyBy()*". By defining the "*initiator*" and "*executor*" as "*address payable*", at run-time this type of EOA allows for exchange of "*Ether*", having access to primitives useful to manage "*Ether*".

The deployment of the contract presented in Listing 4.1, takes 6 seconds and costs 0.000646 "*Ether*" (0,43 $). This contract only offers access to the following variables: *C_facts*, *initiator*, and *executor*. This contract doesn't give access to any function. Meaning, this contract encapsulates the DEMO concepts, but by itself, it is not very useful as it has no business logic.

Smart Contracts can store data and attest transaction execution. From the *Action Model*, it's possible to retrieve which object classes the transaction needs and at which point they arise at the transaction execution, besides evaluating the changes of the objects associated with the transaction execution.

Contract internal state variables represent object classes, with the corresponding DEMO name. The smart contract then serves as a database for the transaction.

To attest transaction execution contracts represent all possible C-facts, according to Figure 2.5. The contract's then hold their current status as C-facts. For every C-act, a contract method exists that changes the contract state to the corresponding C-fact. Every change of C-fact issues an Ethereum system-wide notification, allowing external systems to keep track of their contracts.

The action rules of the *Action Model* define the operations for each actor role. The contract execution logic results from the translation of action rule pseudo-code. The name of the functions correspondent of each C-act is the C-act concatenated with the transaction name.

It is important to note, however, that in the solution presented human actors are ultimately responsible and accountable for the acts of these artifacts. Human actors send transactions to a Blockchain via specific wallet software. The transaction contains a data payload containing instructions to invoke smart contract functions with specific input parameters. The function may update state variables, which are stored, in the contract's internal storage. Each *declare* function call may emit an event, to which an actor can subscribe. All other DEMO transaction steps, if successful, are also recorded in the Blockchain.

Following the nomenclature suggested in [3], this solution does not remove the final responsibility and accountability for the acts from subjects. By subjects, it is understood, human beings responsible for actor roles. The technological capabilities of the Blockchain, however, outperform, in order of magnitude, the subject's capabilities. This dissertation proves the feasibility of implementing actor roles using Blockchain. So, the question is, what it means to have agents perform the P-acts and C-facts while keeping in mind that agents cannot be held accountable. By agents understand Blockchain artifacts to perform functions of, an actor role.

Transactions typically occur on one or more of the *performa*, *informa*, and *forma* levels previously explained in Chapter 2. *Datalogical* transactions, in view of this work, do not present any contradiction in being executed by agents. However, there must always exist an actor ultimately responsible for the work and who can be held accountable for the agent. The solution presented makes the execution of P-acts tacit when declaring a transaction. Since C-facts represent social commitments, C-acts are performed by the actor, himself through the invocation of a Smart Contract function. *Infological* transactions, in the perspective of this work, follow the same guideline as *Datalogical* transactions. The guideline proposed for *Datalogical* transactions allows exchanges of information or knowledge between actors. When it comes to performing original P-acts, at an *Ontological* transaction level, agents are not capable of dealing autonomously with it, as they cannot be ultimately responsible and accountable for the acts. However, Blockchain can support O-actors with the P-acts by doing so as tacitly as possible, that is, by doing so by invoking the function that represents the declaration C-act. However, this form of operation must obviously be known to the actor. This presented solution allows supporting O-actors, to

a large extent, while never taking over the authority and responsibility that is assigned to an actor. This dissertation defends the idea of co-existence between the subjects and the blockchain.

The sub-transactions are implemented as other contracts. In DEMO, there are two kinds of dependencies between BP: *(i)* Request after Promise, request of a new BP is triggered after a promise of a previous; and *(ii)* Request after Accept, request of the new BP is only triggered when the previous one has been accepted. The creation of the sub-contracts inside the enclosing contract at the promise and accept methods respectively solves the issue. The deploying of the sub-contracts must happen at the address returned by the functions.

5

Use Case: Rent-A-Car

Contents

The present chapter covers the Demonstration and Evaluation phases of the DSRM introduced in Section 1.4. It also serves as semantic validation of conceptual mapping presented on Section 4.6.

5.1 Contextualization

The first Use Case is well-known in EO. The case Rent-A-Car is an exercise in producing the essential model of an enterprise that offers the usufruct of tangible things: Rent-A-Car is a company that rents cars to customers. At [3] all four aspect models (*Cooperation Model*, *Action Model*, *Process Model*, and *Fact Model*) are presented. Together they constitute a coherent whole that offers full insight into and overview of the essence of car rental companies.

For a more pragmatic and holistic view of the Rent-A-Car BP, see Figure 5.1.

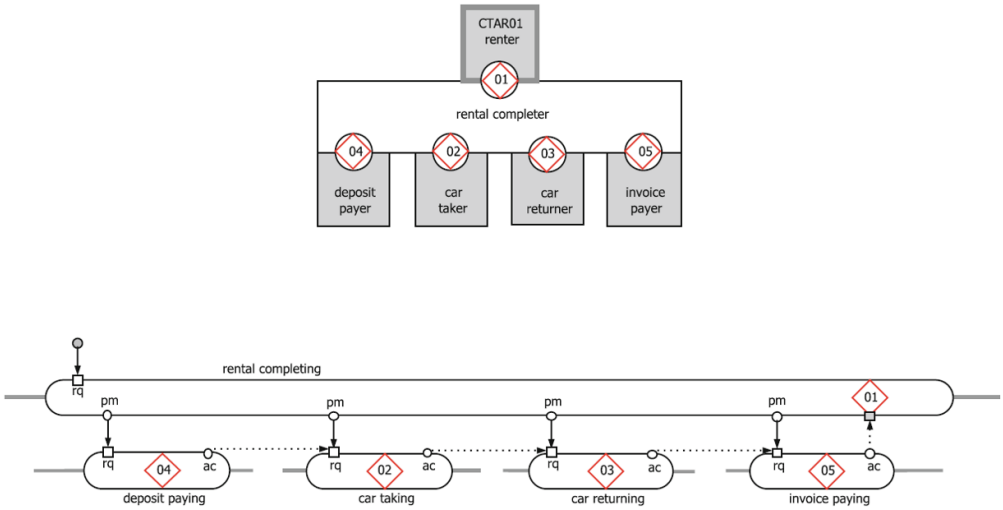

Figure 5.1: Process Structure Diagram (representation form of the *Process Mode*) of the car rental process. Image retrieved from [3].

The BP presented is composed of 5 transaction kinds, the rental completing (T01), that contains the remaining ones, the car taking (T02), the car returning (T03), the deposit paying (T04), and the invoice paying (T05).

The wait link from (T04/ac) to (T02/rq) is a matter of policy of Rent-A-Car. It says that the deposit must have been paid before the car of the rental can be taken. The wait links from (T02/ac) to (T03/rq) and from (T03/ac) to (T05/rq) are a matter of (logistic) necessity. The first says that a car must be taken before it can be returned. The second wait link means that the car of a rental must have been returned before the final invoice can be prepared and be requested to pay.

As an example, the ARS-4, is shown in Figure 5.2. In this case, the event to settle is the car taking of a rental being declared (TK02/da). The response is either an accept (TK02/ac) or a reject (TK02/rj).

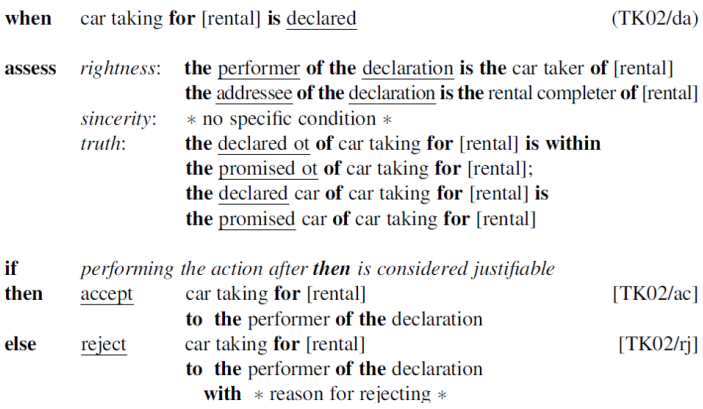

Figure 5.2: Action Rule Specification-4. Image retrieved from [3].

5.2 Implementation

For Rent-A-Car BPs there are 5 consequent transactions with the "*rental completing*" being the parent transaction. The "*rental completing*" transaction can be looked at as the contract between the Rent-A-Car and the Client with conditions that must be fulfilled for the contract to be terminated. These transactions require data sharing between parties as well as trustless control, for this reason, seems fit to implement them resorting to Smart Contracts.

First, the tangling between all the transactions in this Use Case is presented, in Table 5.1, for easier comparison and an execution overview, resorting only to the Action Rules identified in [3] for the Rent-A-Car BP.

```
1  contract RentalCompleting is Transaction {
2      struct Rental {
3          uint256 stratingDate;
4          uint256 endingDate;
5          uint256 maxRentalDuration;
6          uint256 drivingLicenseExpirationDay; ...}
7      //other defined facts
8      Rental public rental;
9      //other declared  facts
10     constructor() public{
11         initiator = 0x7013205512Fd2A988dAA441630E933f5a9A088C9; //rentACar
12         executor = msg.sender; //client
13         rental.maxRentalDuration = 10;
14         //other initialized facts }
```

Listing 5.1: RentalCompleting Contract.

The idea would be for the Client to deploy the "*RentalCompleting*" Smart Contract into the Blockchain

Table 5.1: Sequence systematization of Rental-A-Car Use Case.

RentalCompleting	
rq	with clause of event part (ARS-1)
pm	
DepositPaying	
rq	with clause of response part (ARS-1)
pm	
da	
ac	truth division of assess part (ARS-2)
CarTaking	
rq	with clause of response part (ARS-3)
pm	
da	
ac	truth division of assess part (ARS-4)
CarReturning	
rq	with clause of response part (ARS-5)
pm	
da	
ac	truth division of assess part (ARS-6)
InvoicePaying	
rq	with clause of response part (ARS-7)
pm	
da	
ac	truth division of response part (ARS-8)
RentalCompleting	
da	
ac	

after that, he would be able to request that same transaction.

The contract "*RentalCompleting*" was created. The state variables "*Rental*" and "*CarGroup*" represent the data needed for the BP. In the constructor of "*RentalCompleting*" the state variables that are premises are declred and the addresses of the actors are set too, Listing 5.1. The rest of the "*RentalCompleting*" contract is composed of functions, that each represent a DEMO transaction step. In the specific case of "*RentalCompleting*" there are: "*requestRentalCompleting()*", "*promiseRentalCompleting*", "*declineRentalCompleting()*", "*declareRentalCompleting()*", "*acceptRentalCompleting()*" and "*rejectRentalCompleting()*".

At the "*requestRentalCompleting*", Listing 5.2, the truth division of the assess part of ARS-1 is implemented through the build-in function "*require()*". After all the required check and initializations the state of the "*RentalCompleting* "contract is changed to "*Requested*" by the "*transitionNext*" modifier.

At the "*promiseRentalCompleting*" the contract "*DepositPaying*" is created, Listing 5.2. The "*DepositPaying*" contract must be deployed at the returned address. Note that in this particular case at the end of the function the state is not updated to "*Promissed*" as this will only be done at the "*acceptInvoicePaying*" funtion of the "*InvoicePaying*" contract as showed in Figure 5.1.

```
1  function requestRentalCompleting(uint256 _startingDate, uint256 _endingDate, uint256
       _drivingLicenseExpirationDay) public
2      atCFact(C_facts.Inital) onlyBy(executor) transitionNext(true) {
3          require(_endingDate >= _startingDate);
4          //checks of truth division }
5  function promiseRentalCompleting() public
6      atCFact(C_facts.Requested) onlyBy(initiator) returns (address) {
7          rental.car=rental.carGroup.freeCars[0];
8          DepositPaying depositPaying = new DepositPaying(address(this));
9          return address(depositPaying); }
```

Listing 5.2: RentalCompleting, transaction steps request and promise.

Only at the "*requestDepositPaying()*" the With clause of the action clause of the response part of ARS-1 is implemented through the build-in "*require()*" function, Listing 5.3.

```
1  contract DepositPaying is Transaction {
2      RentalCompleting rentalCompleting;
3      //constructor
4      function requestDepositPaying(uint256 _rq_depositAmount) public
5          atCFact(C_facts.Inital) onlyBy(executor) {
6              RentalCompleting.CarGroup memory cG = rentalCompleting.rental();
7              require(_rq_depositAmount == cG.standardDepositAmount);
8              c_fact = C_facts.Requested; }
```

Listing 5.3: DepositPaying, transaction step request.

After the "*promiseDepositPaying()*", the "*declareDepositPaying()*" implements once again a common pattern in solidity, the Withdrawal from Contracts as this is the recommended method of sending funds. Only the "*declareDepositPaying()*" and "*declareInvoicePaying()*" implement this pattern. Also note that this must be a "*payable*" function and "*p-act()*" modifier must be present.

```
1  function declareDepositPaying() public payable
2      atCFact(C_facts.Promissed) onlyBy(initiator) p_act() transitionNext(true){
3          ( , , , , , , , RentalCompleting.CarGroup memory carGroup) = rentalCompleting
               .rental();
4          require(msg.value >= carGroup.standardDepositAmount);
5          executor.transfer(carGroup.standardDepositAmount);
6          da_depositAmount = carGroup.standardDepositAmount; }
```

Listing 5.4: DepositPaying, transaction step declare.

In DEMO there are two kinds of dependencies between BP: *(i)* Request after Promise, request of a new BP is triggered after a promise of a previous; and *(ii)* Request after Accept, request of the new BP is only triggered when the previous one has been accepted. These two types of dependencies are present in this Rent-A-Car BP, as seen in Figure 5.1.

The first dependency can be seen between the "*promiseRentalCompleting()*" and "*requestDepositPaying()*" functions. The implement of this dependency occurs at the "*promiseRentalCompleting()*" function, where the "*DepositPaying*" Contract is created, having an associated and unique address as seen in Listing 5.2. After this the deployment of the "*DepositPaying*" Contract can be deployed at this same address.

The second dependency can be seen between the "*acceptDepositPaying()*" and "*requestCarTaking()*" functions. This dependecy mimics the end of the "*DepositPaying*" contract lifecycle and the begining of the "*CarTaking*" contract lifecycle. Once again, the "*acceptDepositPaying()*" is where the "*CarTaking*" Contract is created, having an associated and unique address. After this the deployment of the "*CarTaking*" Contract can de deployed at this same address. However, at the "*declareRentalCompleting()*" function, its required to input the address of the "*InvoicePaying*" Contract to prove its existen and check if its the correct instance through the built-in "*require()*" function. For the complete code please go to https://github.com/martasaparicio/try.

5.3 Simulation

In this section, the execution simulation of some steps of the Rent-A-Car BP Contracts, in Remix (Section 4.4.3), are shown. The creating of two addresses is required in Metamask (Section 4.4.4).

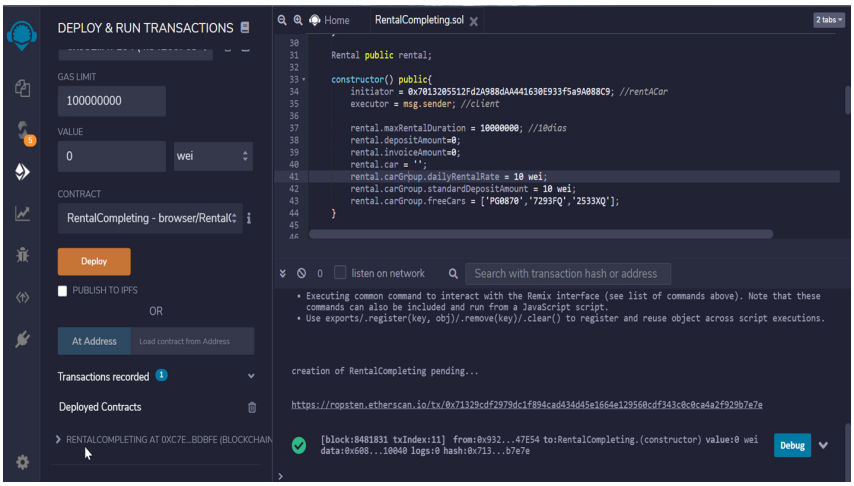

Figure 5.3: Deployment of the RentalCompleting Contract.

The first step is to deploy the "*RentalCompleting*" Controntract into the blockchain. Select the ad-

dress to deploy from that will correspond to the client's address and set the gas limit to 100000000. In Figure 5.3 the contract was deployed, its public variables and functions are listed inside the tab labeled RENTALCOMPLETING on the right side.

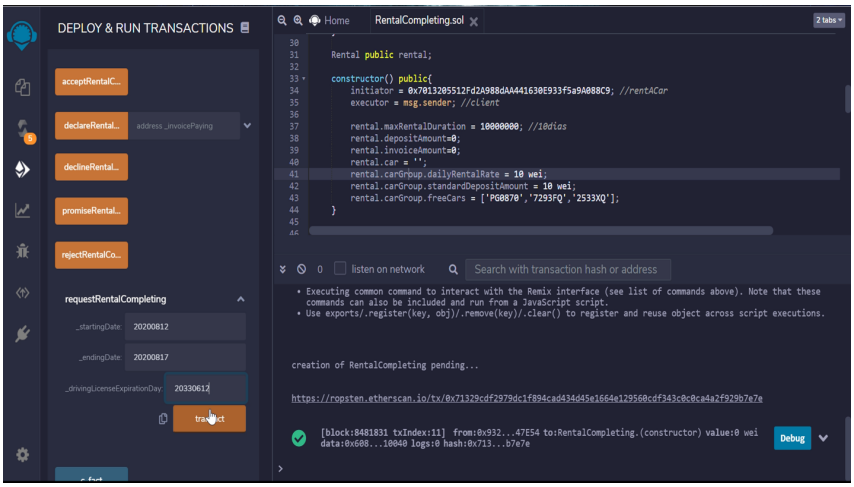

Figure 5.4: Requesting the Rental Completing.

The insertion of input values from the participants in the BP is always done as, shown in Figure 5.4. This figure is showing the insertion data on behalf of the client for the request step.

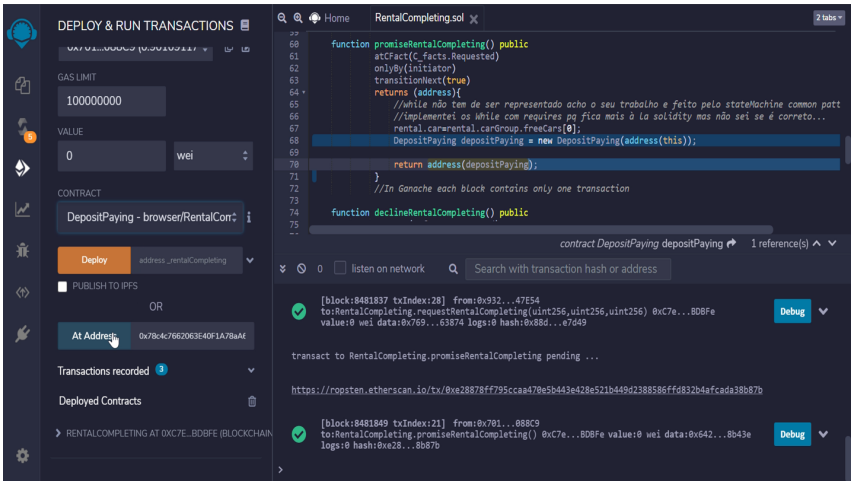

Figure 5.5: Deployment of the DepositPaying Contract.

Time now to see how to create the "*DepositPaying*" Contract. From the execution of the "*promiseRentalCompleting()*" function, on the behalf of the Rent-A-Car, an address is returned. To deploy the "*DepositPaying*" Contract, that retured address must be inserted on the tab "*At Address*", and the file corresponding to the "*DepositPaying*" Contract must be selected from the ones available at the "*CONTRACT*" tab, as showned in Figure 5.5. After this a new tab corresponding to the "*DepositPaying*" Contract will appear below the the correspondent to the "*RentalCompleting*" Contract.

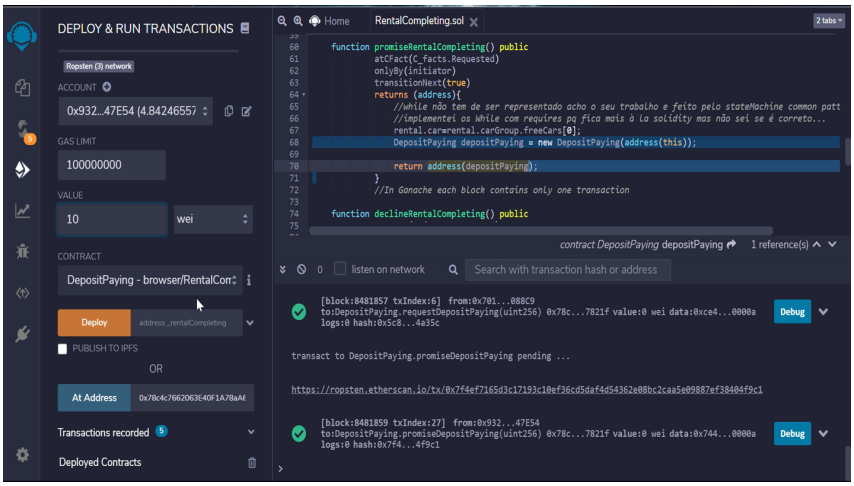

Figure 5.6: Declaring the DepositPaying.

As a last note, please bear in mind that the funds exchanged between addresses must be inserted on the "*VALUE*" tab on the right. The example shown on the Figure 5.6, is relative to 10 wei, that is the deposit required by the Rent-A-Car.

5.4 Evaluation

The evaluation method chosen to validate the artifacts was the Descriptive (simulating scenarios to evaluate the artifacts), suggested by DSRM framework [1]. In this section, an evaluation of the implementation presented in Section 5.2 is developed. From a performance perspective, as well as from a functional argumentation of asset control.

5.4.1 Performance

It's difficult to make a technical evaluation of this process as the blockchain used is the Ropsten test-net, as mentioned in Section 4.4. Although Ropsten test-net is the best test-net that reproduces the current production environment, i.e. system and network conditions on the live Ethereum main-net, because it's Proof-of-Work net, it doesn't allow to change the Block Size for a deeper technological analysis. However, makes available a link to [1] analyze the blocks and their properties. The blocks in this Blockchain have mutable size, being the maximum block size 139 789 bytes and the minimum block size 517 bytes.

A good network is one where an optimal block size for containing as many transaction as possible, while still being able to broadcast that block efficiently to the rest of the network. In Ethereum there

[1] https://ropsten.etherscan.io/

is no fixed block size. Instead, miners vote on a parameter known as gas limit, which is essentially a measure of the computation needed to verify a block. A higher gas limit implies more transactions, and thus a larger block size. This factor is constantly adjusted by ethereum miners based on how efficiently blocks are being propagated at the current size. The gas limit is a cap on both processing and storage/bandwidth because the cost of a transaction/function is fixed in units of gas for each type of instruction. Its also safe to conclude that a lower block size results in a higher throughput.

Figure 5.7: Relation between a sample of block size and latency.

Figure 5.7 figure shows the expected behavior, for a sample of transactions. The larger the block, the shorter the latency of a transaction. This behavior is as expected due to the nature of the chosen Blockchain. In Ethereum the block size dynamically increases if it starts getting full, and decreases if it starts getting empty. This means that when the block size is greater, the number of transactions submitted is also greater, so there is dynamically an increase in the size of the blocks so that the latency does not increase proportionally.

The deployment of the "*RentalCompleting*" Contract was the operation that took the longest to perform, having a latency of 55 seconds. This behavior is expected as this is the contract that contains all the others.

Regarding the transaction step "*request*", Figure 5.8, the "*request*" from the "*Deposit Paying*" transaction was the longest. This may be explained by what was already mentioned. The block size was 2.846 bytes, the smallest found while performing the simulation. This may also be related to the computational capabilities of the miner.

The "*promiseRentalCompleting()*" transaction step was expected to take the longest as it created the address for the "*DepositPaying*" Contract. Although is the second, the one that took the longest was the "*promiseCarTaking()*", this may happened due to the block size difference, 9.310 bytes and 24.716 bytes, respectively.

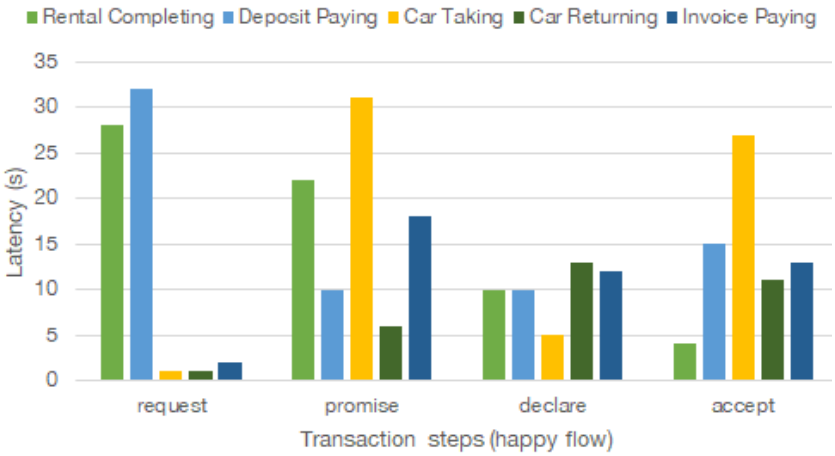

Figure 5.8: Transaction latency for each DEMO transaction step.

On one hand, the "*declare*" transaction step was on average the step that took less time to be mined, 10 seconds. This may be related to the fact that "*declareDepositPaying()*" and "*declareInvoicePaying()*" use buid-in functions for fund transfer.

On the other hand, the "*accept*" transaction step was on average the step that took more time to be mined, 70 seconds. This may be related to the fact that the "*acceptDepositPaying()*", "*acceptCarTaking()*" and "*acceptCarReturning*" create the addresses to other contracts.

Concluding, the latency of these transactions is predominantly determined by the levels of supply, meaning the miners, and demand in the network.

The execution of this Use Case in the Blockchain took on average 326 seconds, which corresponds to 5.43 minutes. Although this solution presents the overhead of translating Action Rules into Solidity Code, this approach allows to reuse Ontological models and guarantees the correct implementation of Smart Contracts.

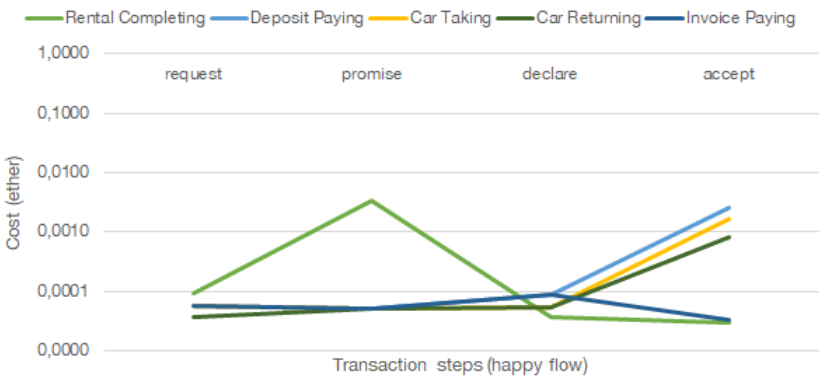

Figure 5.9: Transaction cost for each DEMO transaction step.

A relevant metric when comparing this solution to others is the overall cost. It is safe to say that the

operation that had the greatest financial burden was the deployment of the "*RentalCompleting*" contract has expected, of 0.004667171 "*Ether*", which corresponds to 2.73 $. It was also observed that the price of a transaction is proportional to the complexity of the same.

In Figure 5.9 it is possible to observe that the most expensive transaction step of the "*RentalCompleting*" Contract is the promise step where the "*DepositPaying*" Contract is created. Following the same logic the most expensive step for the contracts "*DepositPaying*", "*CarTaking*" and "*CarReturning*" is the accept step where the subsequent transactions are created. For the customer this process has the cost of 0.005273916 "*Ether*", approximately 3.08 $. For the Rent-A-Car company has the cost of 0.008753809 "*Ether*", approximately 5,12 $. The process grand total is 0.014027725 "*Ether*", which can be converted to 8.20 $.

Three final notes are mandatory to this last analysis. The first being the "*Ether*" values presented in the last paragraph don't really correspond to any value in dollar. The blockchain used was the Ropsten test-net, which means the values exchanged between accounts and used to execute the operations don't have any real value as explained in Section 4.4. The second note goes to the fact that the values presented in the last paragraph vary due to exchange rate volatility. The last relates to the formula used to calculate the transaction cost in the Ropsten test-net. The transaction cost is calculated by the multiplication between the gas price and gas used. Where the gas price corresponds to the cost per unit of gas specified for the transaction, in "*Ether*". The higher the gas price the higher the chance of getting included in a block. The gas used refers to the exact units of gas that were used for the transaction.

5.4.2 Functional Argumentation of Asset Control

As suggested in Section 4.5, and latter on implemented in Section 5.2 each transaction on the Blockchain technology correspondes to a BP step of the DEMO standard pattern [3].

To complete a DEMO BP kind, it is necessary to execute a set of blockchain transactions, meaning a set of BP steps to complete the transaction pattern. For this reason, to complete an inter-organizational scenario, as Section 5.1 shows, this approach implies the submission of a very large number of transactions. It is necessary to submit, in a happy flow case, a total of 20 transactions, not counting the deployment of 1 contract and four other tacitly, to complete the scenario of Section 5.1.

Despite requiring a high number of submitted transactions, this implementation can objectively detect unexpected situations. For instance, an unwanted party tries to take the car that someone else paid for. This is possible due to the use of Blockchain, this technology allows not only to keep track of all transactions but to whom they belong, thus facilitating asset control.

This Use Case presented may ease access to car renting to the different parties involved, especially the information exchange. Furthermore, this Use Case can benefit from the use of blockchain to provide an immutable trace of registry changes. The use of blockchain may also provide higher resilience to

system faults and a more seamless exchange of funds.

Going even further, some ontological transactions could disappear, at least the ones that only exist to deal with truss issues between parties.

In this particular case, the "*CarTaking*" and "*CarReturning*" would be done tacitly, as the access to the asset, in this case a car, would be controlled by the Blockchain. The control could be done through a publish subscribe mechanism. When the client invokes the functions "*declareCarTaking()*" and "*declareCarReturning()*", that consequently invoke the modifier "*p_fact()*", that in turn issues an event representing the "*p_act()*". The Rent-A-Car may listen to these events by subscribing to them.

More generically, the execution of the functions representing the "*declare*" transaction step emit events, which are stored in the transactions tier of the blocks the transactions are contained in. All the other transaction steps are also recorded in the Blockchain, but without the emission of an representative event.

This implementation raises the previously mentioned issue of the disappearance of some transactions that can be tacitly accomplished. Namely, the treatment of sub-transactions should be studied. In this work the the sub-transaction is implemented as another contract, this is not an optimal solution, due to the number of resources that this implies. It is believed there must not always be a need to create separate contract for sub-transactions, the sub-transaction can be implemented inside the main contract. This can be convenient if only partial execution benefits from its Blockchain execution. Finally, the last option is that the sub-transactions are not handled at all and leave this outside of Blockchain. The choice of which option to choose seems at all related to the situation in question. Moreover, it might not be possible to implement full business logic and there is no need to run the exact same transaction execution multiplied on thousands of computers.

There are plenty of works to compare this Use Case with [79–81]. Many of them only focus on the implementation of business rules never thinking on how to improve the process in a pragmatic perspective. The solution presented through a distributed ledger and shared business logic in the form of Smart Contracts, allows enterprises to execute shared business logic and monitor the state of process instances [12].

6

Use Case: European Union Parliament Elections

Contents

The present chapter covers the Demonstration and Evaluation phases of the DSRM introduced in Section 1.4. It also serves as semantic validation of conceptual mapping presented on Section 4.6.

6.1 Contextualization

Since 1979, occurred to date, nine Elections of the European Parliament, once every five years according to the universal adult suffrage. By 2020, 704 Members of the European Parliament will be elected by more than 400 million eligible voters from 27 member states. For this reason, it is considered the second-largest democratic election in the world [82]. Each member state has its voting system, either being by Preferential Voting, Closed Lists, or Single Transferable Vote, all can be combined with Multiple Constituents. Further, each member state has its voting methods for citizens resident abroad and if whether or not voting is compulsory. Each member state has different rules determining who can vote for and run as a Member of the European Parliament. Although this is not a very standardized process, the European Parliament is the only institution in the European Union, directly elected by the citizens. So, it is extremely important that this is a transparent, meddle-proof, usable, authenticated, accurate, and verifiable process [83].

The development and implementation of constitutional and legal provisions have been one of the engineering concerns. Since for instance, the Elections of the European Parliament have an estimated cost €700 million. Blockchain has been the most promising technology when it comes to the electoral domain, seeing that per each vote, a new transaction, if valid, is added to the end of the blockchain and remains there forever [84]. For this solution, no centralized authority is needed to approve the votes, and everyone agrees on the final tally as they can count the votes themselves, as anyone can verify that no votes were tampered with and no illegitimate votes were inserted.

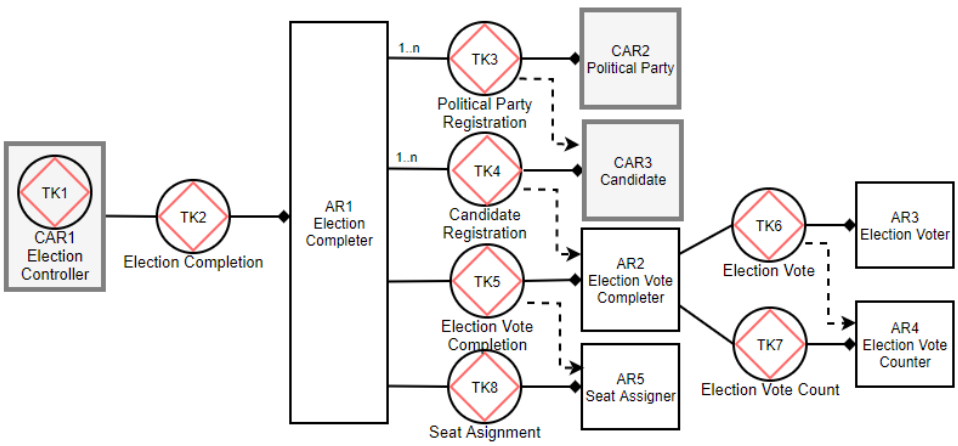

Figure 6.1: Process Structure Diagram (representation form of the *Process Mode*) of the European Union Parliament Elections process.

The European Union election process is very complex. The European Union issues general guidelines on how country elections should look like, and each country then implements its legislation to describe how the elections are done in a particular country. This means that each of the 27 European Union countries do this process differently. To avoid this complexity, it was assumed that there is a unified voting process and each country votes according to one of the three voting systems – Preferential Voting, Closed Lists, and Single Transferable Vote.

Once again, for a holistic view over the BP considered, the Figure 6.1 shows the Process Structure Diagram, representation form of the *Process Mode*. The process starts with a "*Election Controller*", which every five years triggeres the process. After that, the "*Election Completer*" is responsable for medianting the remaining transactions.

6.2 Implementation

Let's start this implementation section by understanding the token concept, that until now went unnoticed. From what has been written throughout this work, the use of Blockchain technology seems the future to manage digital assets, due to its security and immutability features. In [85] tokens are described as ways to represent "*programmable assets or access rights, managed by a smart contract and an underlying distributed ledger. They are accessible only by the person who has the private key for that address and can only be signed using this private key*" [85].

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

Due to the need to standardize the general structure of Ethereum Blockchain tokens, Ethereum Improvement Proposals [1] was created. These standards allowed for Smart Contracts running on Ethereum to interact with tokens defined by other Smart Contracts. One of these standards is the ERC-20, providing a typical list of rules on how the fungible tokens should be structured. That makes them easy to trade, as there is no need to differentiate the tokens. To represent unique assets, however, the tokens must be non-fungible, even if they are of the same type. In order to facilitate these types of tokens, the ERC-721 standard was created [85].

[1] https://eips.ethereum.org/

```
1  import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v3.1.0/
       contracts/access/Ownable.sol";
2  import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v3.1.0/
       contracts/token/ERC721/ERC721.sol";
3  contract VotingToken is Ownable, ERC721{
4    constructor(string memory name, string memory symbol) ERC721(name, symbol) Ownable
         () public payable{  }
5    function mint(address receiver) onlyOwner public {
6      _safeMint(receiver, uint256(receiver)); }
7    function transfer(address from, address to) onlyOwner public {
8      _transfer(from, to, uint256(from)); } }
```

Listing 6.1: DepositPaying, transaction step declare.

Due to the requirement of using non-fungible tokens for this Use Case, a "*VotingToken*" contract was created, as shown in Listing 6.1.

```
1  contract ElectionCompleting is Transaction{
2    struct CountryElections{
3      uint256 id;
4      string countryName;
5      uint electionBeginDate;
6      uint electionEndDate;
7      address[] voters;
8      VotingSystem votingSystem;
9      int availableSeats;
10     uint8 electoralTreshold;
11     uint8 minimumAge; }
12   //other  defined  facts
13   VotingToken votingToken;
14   CountryElections[] countries;
15     //other  declared   facts
16    constructor() public payable{
17     votingToken = new VotingToken("VotingToken","VOTE");
18     CountryElections memory portugalElections = CountryElections(1,"Portual",
           20210124, 20210124, new address[](0), VotingSystem.ClosedList, 21, 0, 18);
19     countries.push(portugalElections);
20     //other initialized facts }
```

Listing 6.2: Election Completing Contract.

About 30 Action Rules were created to implement the process shown in Figure 6.1, available at https://github.com/martasaparicio/try. The idea would be for the end result of "*ElectionControl-*

ling" to result in the deployment of the "*ElectionCompleting*" Smart Contract into the Blockchain, Listing 6.2, after that, it would be possible to request the registration of Political Parties.

Although more complex, the chaining of transactions and contracts would process in a similar way to what was exposed in the Chapter 5. For this reason, this Use Case will have a greater focus on the logic of the Use Case, as it is supposed when deriving the contracts from the Transaction Contract (Listing 4.1).

```
1  function declarePoliticalPartyRegistring(string memory name, string memory code,
       string memory website, uint256 countryId) public
2      atCFact(C_facts.Promissed) onlyBy(initiator) p_act() transitionNext(true) {
3          require(politicalParties[msg.sender].id == address(0));
4          //other restrictions
5          PoliticalParty memory party = PoliticalParty({id: msg.sender, name: name,
               code: code, website: website, voteCount: 0, allocatedSeats: 0, countryId:
               countryId});
6          politicalParties[msg.sender] = party;
7          politicalPartiesKeys.push(msg.sender); }
```

Listing 6.3: Political Party Registering, transaction step declare.

The execution of the registration of a political party, occurs in the function "*declarePoliticalPartyRegistering()*", in the contract "*PoliticalPartyRegistering*", as can be seen in Listing 6.3. Please note that the modifier "*p_act()*", is only used in functions that implement the "*declare*" DEMO transaction step. This modifier, as shown in Listing 4.1, issues an event correspondent to a unique "*p_fact()*".

```
1  function declareCandidateRegistring(address partyId, string memory name, string
       memory website) public
2      atCFact(C_facts.Promissed) onlyBy(initiator) p_act() transitionNext(true) {
3      PoliticalParty storage party = politicalParties[partyId];
4      require(bytes(party.name).length > 0);
5      require(candidates[msg.sender].id == address(0));
6      Candidate memory candidate = Candidate({id: msg.sender, name: name, website:
           website, voteCount: 0, hasSeat: false, approved: true, partyId: party.id});
7      candidatesKeys.push(msg.sender);
8      candidates[msg.sender] = candidate; }
```

Listing 6.4: CandidateRegistration, transaction step declare.

The realization of the "*CandidateRegistering*" transaction is similar to the realization of the "*PoliticalPartyRegistering*" (Listing 6.4).

```
1  function promiseElectionVoteCompleting(uint256 countryId) public
2      atCFact(C_facts.Requested) onlyBy(initiator) transitionNext(true) returns (
           address){
3          CountryElections storage country = countries[countryId];
4          for (uint256 i = 0; i < country.voters.length; i++) {~
5              votingToken.mint(country.voters[i]);}
6          ElectionVoteCounting electionVoteCounting = new ElectionVoteCounting(address(
               this));
7          return address(electionVoteCounting); }
```

Listing 6.5: ElectionVoteCompleting, transaction step promise.

A very important operation for the "*votingToken*" is the "*mint()*" operation, at line 5 of Listing 6.5, as a token only start to exist after its activation by the "*mint()*" operation. This operation in done by the "*Election Vote Completer*", respecting the ERC721.

```
1  function declareElectionVoting(uint256 countryId, address[] memory votingchoices)
       public atCFact(C_facts.Promissed) onlyBy(initiator) p_act() transitionNext(true){
2        require(votingchoices.length > 0, "At least one cadidate must be chosen");
3        CountryElections storage country = countries[countryId];
4        require(now > country.electionBeginDate, "Voting is currently not allowed");
5        if (country.votingSystem == VotingSystem.ClosedList) {
6              require(votingChoices.length == 1, "Only a party vote is allowed");
7              PoliticalParty storage party = politicalParties[votingChoices[0]];
8              require(party.id != address(0), "Party address is invalid");
9              require(party.countryId == countryId); }
10         else if (country.votingSystem == VotingSystem.OpenList) {
11             address firstCandidateParty;
12             bool first = true;
13             for (uint i = 0; i<votingChoices.length; i++){
14                 require(candidates[votingChoices[i]].approved==true, "Candidate
                       address is invalid");
15                 address currCandidateParty = candidates[votingChoices[i]].partyId;
16                 if(first) {
17                     firstCandidateParty = currCandidateParty;
18                     require(politicalParties[firstCandidateParty].countryId ==
                           countryId);
19                     first = false; }
20                 else {
21                     require(currCandidateParty == firstCandidateParty, "Candidates
                           must be from the same party"); } } }
22         else if (country.votingSystem == VotingSystem.SingleTransferable) {
23             for (uint i = 0; i<votingChoices.length; i++){
```

```
24                  Candidate storage candidate = candidates[votingChoices[i]];
25                  require(candidate.approved == true, "Candidate address is invalid");
26                  require(politicalParties[candidate.partyId].countryId == countryId);
                        } }
27          votingToken.transfer(msg.sender, address(this));
28          if (country.votingSystem == VotingSystem.ClosedList){
29              PoliticalParty storage party = politicalParties[votingchoices[0]];
30              party.voteCount += 1; }
31          else{
32              for(uint i = 0; i<votingchoices.length; i++){
33                  Candidate storage candidate = candidates[votingchoices[i]];
34                  candidate.voteCount += 1; } } }
```

Listing 6.6: ElectionVoting, transaction step declare.

"*declareElectionVoting()*", Listing 6.6, contains a more complex logic. For lines 2 to 4 general validations, are being made. From lines 5 to 9, the votes from countries with Closed List as a voting system are being validated. From lines 10 to 21, the votes from countries with Open List as a voting system are being validated. From lines 22 to 26, the votes from countries with Single Transferable as a voting system are being validated. At line 27, the "*votingToken*" is being transferred from "*msg.sender*" to "*address(this)*" according to ERC721. At last, the votes are being counted depending on the voting system adopted per country. If a country follows the Closed List system, lines from 28 to 30 are executed otherwise, lines from 31 to 34 execute.

The address of a voting citizen and his voting ballot are not the parameters to prevent people from acting as someone else. The citizen's id is taken from the "*msg*" object that contains a verified public key by the original sender. After the validations, a vote is counted according to the country voting system and the citizen's vote is transferred back to the smart contract to prevent a double vote. Note that the casting vote is counted but not forgotten due to Blockchains' inherent history keeping. The Blockchain also guarantees that the function is processed one at a time. Lines 30 and 34 are safe to use even when multiple sources in parallel call them.

```
1  function declareElectionVoteCounting(address partyAddress, address[] memory
       candidateAddresses) public
2      atCFact(C_facts.Promissed) onlyBy(initiator) p_act() transitionNext(true){
3          if (country.votingSystem == VotingSystem.ClosedList){
4              PoliticalParty storage party = politicalParties[partyAddress];
5              da_voteCount = party.voteCount; }
6          else{
7              for(uint i = 0; i<candidateAddresses.length; i++){
8                  Candidate storage candidate = candidates[candidateAddresses[i]];
```

```
9                   da_voteCount =candidate.voteCount;} } }
```

Listing 6.7: ElectionVoteCounting, transaction step declare.

The vote counting procedure is done tacitly, Listing 6.7. If it is a country that implements a Closed List system then vote count is already stored at the party vote count, if the country implements Preferential Voting or Single Transferable Vote the count will be stored at candidate vote count.

The process of assigning seats to each candidate is very complex as depends on the voting system of each country, in particular, Preferential Voting and Single Transferable Vote are complex algorithms. The resulting code is very extensive and complex for this reason this the implementation of this DEMO transaction can only be found at `https://github.com/martasaparicio/try`. All the remaing contracts can also be found in the referred repository.

6.3 Simulation

In this section, the execution simulation of some steps of the European Union Parliament Elections BP Contracts, in Remix (Section 4.4.3), are shown. Metamask (Section 4.4.4) was used again to create the desired number of EOAs.

The use of IDE Remix (Section 4.4.3) at all resembles that presented in Section 5.3. The compilation of contracts, their deployment, the insertion of input, and the collection of output values.

Note, however, that the "*VotingToken*" contract that represents the voting token (Listing 6.1) represent's an overhead when comparing this solution to the one presented in the last chapter. Nonetheless, this overhead is essential to represent unique assets, which can only be achieved by non-fungible tokens.

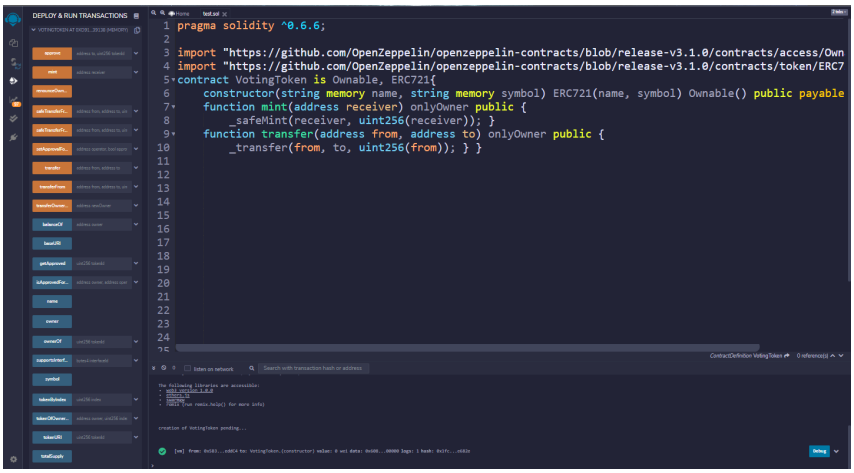

Figure 6.2: Deploying Voting Token Contract.

When deploying the "*VotingToken*" contract, Figure 6.2, by extending "*Ownable*" and "*ERC721*", several methods are inherited. However, for the execution of this Use Case, only the following methods are relevant: "*mint()*" and "transfer()". The first, marks the beginning of the life cycle of a token. The second, transferring the token between addresses.

In section Section 5.3 of the last chapter, it was demonstrated how to deploy a contract in Figure 5.3; insert input values in Figure 5.4; deal with the two types of dependencies possible between BPs in DEMO (request after promise and request after accept) in Figure 5.5. Although it is not necessary for this BP to exchange any funds between addresses, that is also simulated in Figure 5.6.

For this reason, in this section, we will address an artifact developed within the scope of this dissertation. This is an artifact that, despite not being presented as one of the main ones, is extremely important and possible future work to develop more completely.

An ontological builder provides the best ontology and knowledge representation practices, together with the best enterprise solutions architecture to provide a robust and scalable ontology management solution. The essential function that is being considered is that an ontological builder translates an ontological model, fully independent of the implementation, into an implementation model, this process being the most straightforward definition of engineering. In software development, the implementation model often is the source code in a programming language like Solidity. None of the ontological builders already on the market seemed to fulfill the needs of this work, going from the ontological model DEMO Action Model, to Solidity code.

Two technologies were considered to implement the first prototype of the ontological builder. The first being Google Blockly [2], a visual coding block editor. Blockly documentation claims that this editor is an intuitive, visual way to build code [3], which is in line with the requirements to model Action Rules. An action rule can be interpreted as a block that contains the normal Action Rules Specification presented as Algorithm 2.1. The second technology considered was Java 2 Enterprise Edition (J2EE) [4]. J2EE is a standard for implementing and deploying enterprise applications. The term "enterprise" implies highly-scalable, highly-available, highly-reliable, highly-secure, transactional, distributed applications. The J2EE technology is designed to support the rigorous demands of largescale, distributed, mission-critical application systems and provides support for multi-tier application architecture [86].

Although the second technology is clearly more widely used, the first seems to be more accessible to a larger number of people. This contribution will mainly affect non-technical people, that do not comprehend software code, and without this knowledge, are once again dependent on a centralized organization to write the contracts.

It took three changes to Blockly technology, but due to its open-source nature, it was possible to

[2] https://developers.google.com/blockly
[3] https://developers.google.com/blockly/guides/overview
[4] https://www.oracle.com/java/technologies/appmodel.html

make it happen. It was necessary to create customized blocks that were in line with the action rules specification already presented Algorithm 2.1. The second change was the production of Solidity code since, Blockly generates only JavaScript, Python, PHP, Lua, Dart, and XML code.

To satisfy these requirements it is necessary to create, for each customized block, a Block definition object, a Toolbox reference, and a Generator function. The Block definition object defines the look and behavior of a block, including the text, color, fields, and connections. Once defined,the type name must be used to reference the block to the toolbox. The toolbox reference allows users to add it to the workspace. Finally, to transform the block into code, the block is paired with a generator function. The generator in question is specific to the desired output language, Solidity. The generator function takes a reference to the block for processing. It renders the inputs into code strings and then concatenates those into an expression.

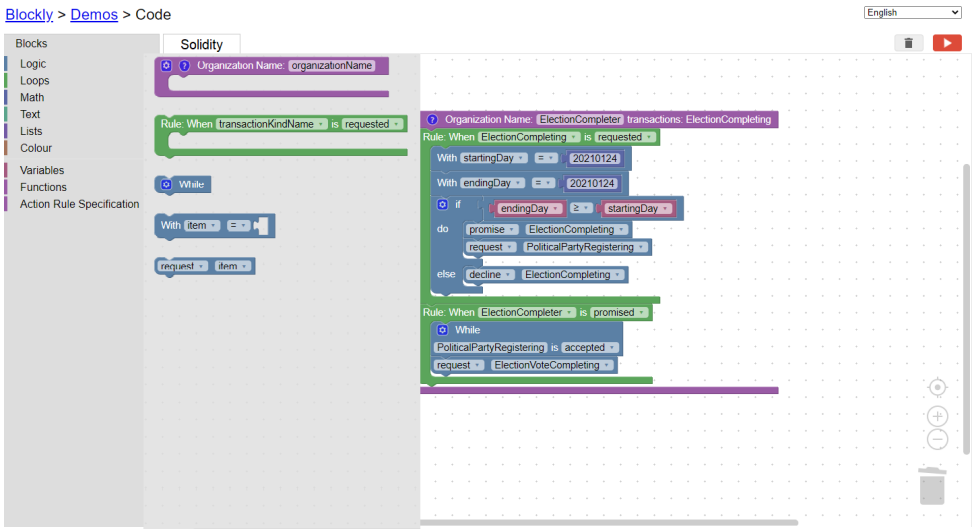

Figure 6.3: Blockly workspace, defining ARS-5 and ARS-6.

As an example the workspace is presented Figure 6.3. In the figure the first two action rules to be performed by the actor "*Election Completer*" are presented.

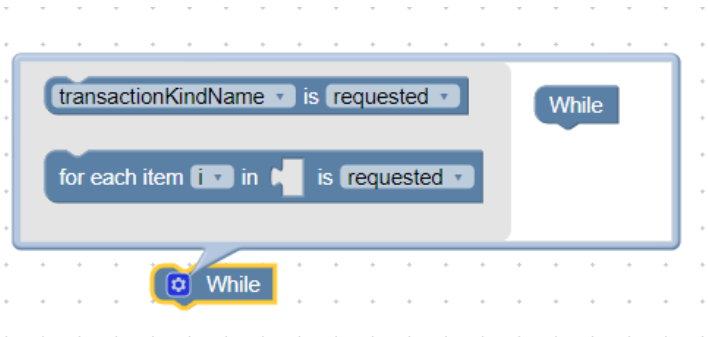

Figure 6.4: while_clause block.

Some of the custom blocks, for example, the "*organization_contract*" block and the "*while_clause*" block use, extensions or mutators to be even more dynamic and configurable. Extensions are custom configuration or behavior for blocks which can be applied to a block through the block's definition. A mutator is very similar to an extension. Mutators are, however, the only way to provide a custom serializable state on a block. An example can be seen in Figure 6.4. The initial prototype of the ontological builder can be found, like the remaining artifacts in https://github.com/martasaparicio/try.

The Ontological Builder, as already mentioned, was not the main objective of the thesis, so its development still has room for improvement. In particular, it would be interesting to explore the technology already mentioned, J2EE, too compare solutions. However, it was possible to create some of the simplest contracts of the use case presented in this chapter.

6.4 Evaluation

The evaluation method chosen to validate the artifacts was the Descriptive (simulating scenarios to evaluate the artifacts), suggested by DSRM framework [1]. In this section, an evaluation of the implementation presented in Section 5.2 is developed. From a performance perspective, as well as from a functional argumentation of asset control.

6.4.1 Performance

As mentioned in Section 5.4, the Blockchain used is the Ropsten test-net. Although Ropsten test-net is the best test-net that reproduces the current production environment, i.e. system and network conditions on the live Ethereum main-net, because it's Proof-of-Work net, it doesn't allow to change the Block Size for a deeper technological analysis.

As discussed in Section 5.4, a good network is one where an optimal block size for containing as many transaction as possible, while still being able to broadcast that block efficiently to the rest of the network.

In Ethereum, there is no fixed block size. Instead, miners vote on a parameter known as the gas limit, which is essentially a measure of the computation needed to verify a block. A higher gas limit implies more transactions and thus a larger block size. This factor is constantly adjusted by Ethereum miners, based on how efficiently blocks are being propagated at the current size.

As proven by Figure 6.5 figure shows the expected behavior, for a sample of transactions. Both variables have an inversely non-proportional relationship. The block size is not a controllable metric, so it is not a metric by which, can extract knowledge beyond which is public domain about the operation of the Blockchain Ethereum, a lower block size results in a higher throughput.

Figure 6.5: Relation between a sample of block size and latency.

As previously explained, this use case requires non-fungible tokens to represent voter's ballots, preventing a double vote situation. This requirement requires additional costs associated with the deployment of the "*VotingToken*" Contract.

The time spent in the deployment of this contract was 14 seconds, although it does not have complex operations, the extension of two other complex contracts, "*Ownable*" and "*ERC721*", justifies this time.

The cost was of 0.022150379098 "*Ether*", which is equivalent, at the time of writing, to 13,08 $. When compared to the total cost of the Use Case in the previous chapter of 8.20 $, can be immediately realized that this Use Case has a much higher cost than the previous. Since the non-fungible token representing only a voter's ballot, costs more than the complete Rent-A-Car process.

Once again, the operational dynamics of Ethereum Blockchain was proven. As already noted, the relationship of the transaction latency for each DEMO transaction step is associated with two variables. The first being the block size, which varies depending on the vote of the miners, in regards to the computation needed to verify a block. A higher value implies more transactions and thus a larger block size. The second variable being the computational capabilities of the miner. Miners validate new transactions and record them on the blockchain. A miner with a higher computational capacity computes more quickly difficult mathematical problems based on cryptographic hash algorithms, required to append a bock to the blockchain.

The trend seen in the previous chapter remains, "*accept*" was on average the transaction step that took more time to mine, 90 seconds. This may be related once more with the sequence between the transactions, since it is in this step that the contract for the next transaction is deployed.

Despite the considerable increase in complexity associated with each "*declare*" transaction step, this remained, on average, the fastest transaction step to mine.

The rest, like those already mentioned, DEMO transaction steps behavior varies according to the

uncontrollable variables already mentioned (block size and miner). This behavior should not be seen as a failure. The main reason for choosing Ropsten test-net, was exactly that it mimicked as closely as possible the behavior of the public blockchain Ethereum.

Once again, the conclusions point that he latency of these transactions is predominantly determined by the levels of supply, meaning the miners, and demand in the network.

The deployment of the "*Election Controlling*" was the operation that took the longest to perform. This behavior is expected as this is the contract that contains all the others.

All transactions showed the expected behavior in Figure 6.6. However, it is worth noting the "*ElectionVoteCompleting*" transaction, which has a behavior not yet seen, but which is in line with expectations.

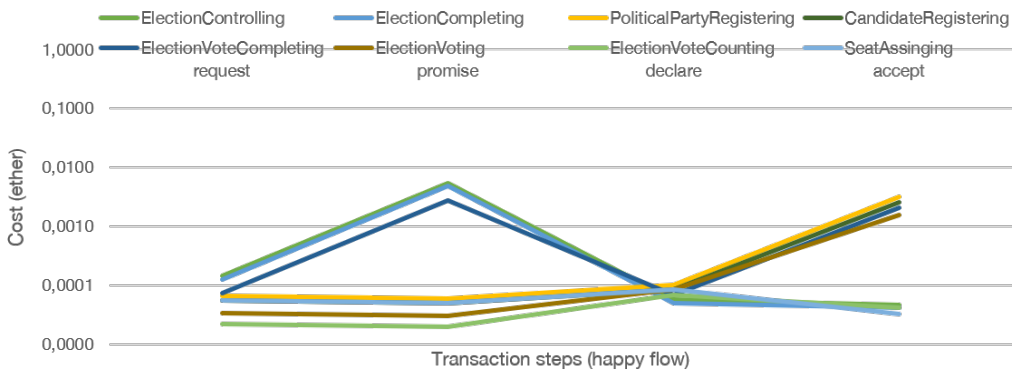

Figure 6.6: Transaction cost for each DEMO transaction step.

Since it contains two other transactions as expected, it has a high cost in the transaction step "*promise*". What distinguishes it from the other two transactions that contain more transactions ("*ElectionComtrolling*" and "*ElectionCompleting*"), is the fact that it has at the end of its life cycle to create the contact corresponding to the "*SeatAssinging*" transaction. Thus, this transaction combines the behaviors that until now had been observed separately, having a high cost in steps "*promise*" and "*accept*".

It was not possible to test this use case with a real number of countries, candidates and voters, due to the complexity that this entailed. However, it is safe to say that the conclusions drawn are the same as those of the previous use case.

Without accounting for the cost and time required to create the ballot for each voter, which has already been presented separately, the total cost of this use case was 0.02244436 "*Ether*", which is equivalent to 15.43 $, the total time was 654 seconds, which equals 10 minutes and 9 seconds.

The limitation of this approach is that mainly there is currently no public blockchain capable of supporting millions of transactions in a cost and time-efficient way, as it would be necessary for this Use Case to come to life. According to Gartner, [87], production-ready blockchain technology will be available by 2030. By then, the transaction fees should be lower, so significant cost savings are expected.

6.4.2 Functional Argumentation of Asset Control

This Use Case is more interesting as well as more complex. In this Use Case, there is a deeper hierarchy of responsibilities than in the previous Use Case. The largest hierarchical chain comprises the Election Controller, the Election Completer, the Election Vote Completer and, for example, the Election Vote Counter. This hierarchy and division of responsibilities facilitates the asset control.

It is important to understand the practical implications that the application of Blockchain technology can have in this and other BPs.

With the application of the Blockchain, it would make sense to implement the "*Election Controlling*" transaction through an Oracle. An Oracle, as mentioned in Section 2.3, is a data feed provided by an external service and designed for use in Smart Contracts on the Blockchain. Oracle provides external data and triggers Smart Contract executions when predefined conditions are met. An Oracle in the context of Blockchains and Smart Contracts is then an agent that finds and verifies real-world occurrences and provides this information to a Blockchain to be used by Smart Contracts. In this particular case, the "*Election Controlling*" transaction would disappear to make way for the implementation of an Oracle, that every five years would trigger the execution of the "*Election Completing*" transaction.

To better understand the control of the assets, in this Use Case, it's important to see them as real-world entities. The "*Election Completer*" actor, in a real-world scenario, is represented by the entity responsible for the coordination of the European Union Parliament Elections in all 27 member states. The "*Election Vote Completer*", on the other hand, is represented by the entity responsible for the coordination of the European Union Parliament Elections in each member state. The access of each of these entities is done through an EOA that allows them to carry out their agenda, following the corresponding action rules. A very essential actor in this process is "*Election Voter*". This actor represents voters in a real voting scenario. The "*Election Voting*" transaction could never be suppressed by the blockchain, since it is based on a single decision that can only be taken by an actor.

But the same does not happen with actors "*Election Vote Counter*" and "*Seat Assigner*". The result of both actor "*Election Vote Counter*" and actor "*Seat Assigner*"'s work takes place in an integrated way in the functionalities of Blockchain technology.

As the votes are submitted by voters, in this case by the "*Election Voter*", their counting is done intrinsically. Which can lead to the disappearance of the "*Election Vote Counting*" transaction. Blockchain technology has the ability to secure and validate the voting process on its own, as it secures a person's vote and doesn't allow any other actor to change its vote. For this solution, no centralized authority is needed to approve the votes, and everyone agrees on the final tally as they can count the votes themselves, as anyone can verify that no votes were tampered with and no illegitimate votes were inserted.

Now the allocation of seats in parliament can also be done automatically using blockchain technology. The blockchain has access to all candidates if the voting system followed is Preferential Voting or Single

Transferable, or all candidates associated with a party if the voting system followed is Closed List. When the assignment is being made, it's possible to assign a token to each of them, now deputies. With this token, they would be able to identify themselves in the course of their duties. For example, voting in parliament would be done using this token. This token would allow the citizens of each country to verify the active participation, or not, of the elected candidates.

The changes would translate into the restructuring shown in Figure 6.7. The benefits of applying Blockchain technology to the use case, such as reducing the number of transactions and asset control, have been described. Now, the use of Ontology provides another type of benefit, provide solid and robust principles, in addition to reusing and organizing knowledge.

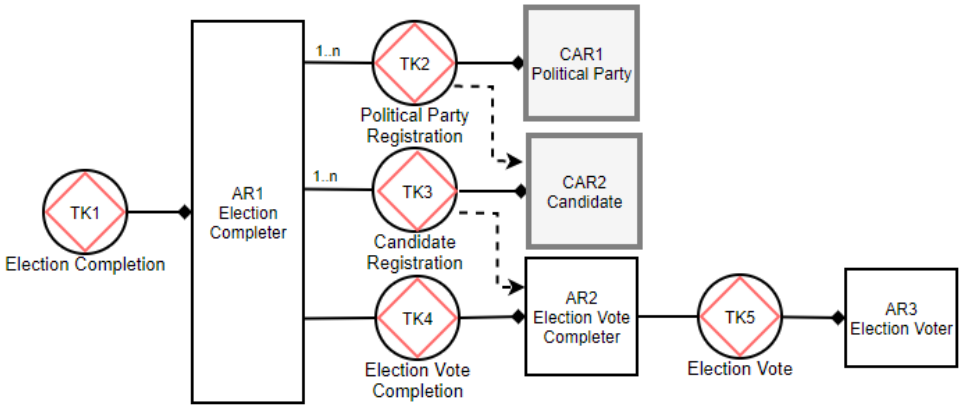

Figure 6.7: Process Structure Diagram (representation form of the *Process Mode*) of the European Union Parliament Elections process using Blockchain.

Moreover, Ontology can be used for communication purposes by ensuring interoperability between computer programs and humans at the data and process level, disambiguate or uniquely identify the meaning of concepts in a given domain of interest, or by facilitating knowledge transfer by excluding unwanted interpretations through the usage of formal semantics [88].

As can be seen was made the option of keeping the "*Election Vote Completer*" actor, for it represents a different real-world entity. This decision is also related to the fact that this actor has different responsibilities than those assigned to the "*Election Completer*" actor. This separation not only facilitates the control of access to votes, which are the assets in question, but also assigns responsibility to each of the actors in the event that something goes wrong. In this way, the actor "*Election Vote Completer*" remains the ultimate responsible for the execution of voting and vote counting in the respective member state.

The limitation of this approach is that mainly there is currently no public blockchain capable of supporting millions of transactions in a cost and time-efficient way, as it would be necessary for this Use Case to come to life. According to Gartner, [87], production-ready blockchain technology will be available by 2030. By then, the transaction fees should be lower, so significant cost savings are expected.

7

Communication

This chapter corresponds to the communication of the importance of the problem, the artifacts, and its utility and novelty to researchers and relevant conferences which constitutes the sixth and final step of DSRM, the methodology adopted for this thesis.

To obtain scientific evaluation about the conceptual mapping developed and presented in Chapter 4 the following paper was submitted and accepted in:

- **Aparício, M**.; Guerreiro, S. and Sousa, P. (2020). **Towards an Automated DEMO Action Model Implementation using Blockchain Smart Contracts.** In Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS)

To obtain scientific evaluation about the Use Case implementation developed and presented in Chapter 5 the following paper was submitted and accepted in:

- **Aparício, M.**; Guerreiro, S. and Sousa, P. (2020). **Automated DEMO Action Model Implementation using Blockchain Smart Contracts.** In Proceedings of the 12th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (KEOD)

To obtain scientific evaluation about the Use Case implementation developed and presented in Chapter 6 the following paper was submitted and accepted in:

- Skotnica, M.; **Aparício, M.**; Pergl, R. and Guerreiro, S. (2021). **Process Digitalization using Blockchain – EU Parliament Elections Case Study.** Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)

8

Conclusion

Contents

In this chapter, a reflection on the developed work and the obtained results are presented. Some general conclusions about the applicability of the solution, including its potential, will be discussed. An outlining of some guidelines for future work are drawn-out.

8.1 Conclusions

In the world of business, organizations aim to sell the best products and services to their clients to provide the maximum value to them efficiently and effectively, reducing costs, and maximize profits. To do so is essential that organizations execute tasks named BPs, which define how the work is organized and coordinated to produce a certain product or service [89].

Blockchain technology can be defined as a decentralized ledger that records transactions and allows the tracking of assets securely and reliably [9]. A blockchain is decentralized, in the sense that it is composed of a set of nodes managed by different entities, so it is not controlled by any entity individually. Blockchain properties are particularly useful in inter-organizational processes where actors do not necessarily trust each other. Blockchain allows for an authentic and confidential exchange between BP parties. Blockchain 2.0 introduces concepts for flexible and programmable transactions referred to as smart contracts.

Over the past two centuries, trade has grown significantly, completely changing the global economy. The integration of the national economy into the global economic system has been one of the most important developments of the last century [10]. Since ancient times, binding agreements have been drawn up to recognize and manage the rights and obligations between parties. These binding agreements, for value exchange, are now called contracts and protect both parties. These contracts are written or spoken and require the parties to trust each other to fulfill their commitments.

A characteristic of collaborative BPs is that they are composed of different sub-processes executed by various organizations. By their nature, sub-processes carried out by one organization are usually beyond the domain of influence of all other collaborators [12]. But in the context of the process, the results of the sub-process may be important to other collaborators. Therefore, from the perspective of all other collaborators, this creates uncertainty in the process.

Blockchain technology can be seen as a solution to coordinate inter-organizational processes involving untrusted parties [8]. This technology provides a way to record something that happened to ensure that it cannot be deleted once recorded. Smart Contracts are mechanisms, that exist in latter Blockchain generations, that ensure that a given routine is executed every time a transaction, of a given type, is recorded. This alternative method ensures that important business rules are always followed.

Through a distributed ledger and shared business logic in the form of Smart Contracts, enterprises can execute shared business logic and monitor the state of process instances [12]. Blockchain allows

organizations to use a single shared database to manage and operate their shared BP.

While Blockchain can guarantee that the stored data cannot be tampered with, it cannot guarantee that the data was correct when it was stored in the chain. Rectifying incorrect information is extremely complex, if not impossible, in a blockchain, and the absence of a central authority makes it difficult to prevent fraudulent entries or hold the fraudster accountable. What is needed is a correctness check for data before data is stored in the blockchain.

Concerning the automatic generation of Smart Contracts, several approaches can be followed, one of which being a MDE approach. In the context of Blockchain applications, MDE is of particular importance as the Blockchain is by design an immutable record, so it is non-trivial or even infeasible to update Smart Contracts.

Enterprises are complex systems involving both human and technological aspects [7]. In DEMO, an enterprise is seen as a system of people and their relations, authority, and responsibility.

The DEMO methodology fulfills the C_4E quality criteria and, in its complexity, offers all the information about the process needed to evaluate and create smart contracts. From the *Action Model*, it is possible to infer transaction states, their order, and names. Furthermore, it's possible to know what information is needed, in each transaction, what are the relations between them, their flow, and action rules.

By computing a high-level formal model as DEMO *Action Model* and generating Smart Contracts from it, one can create smart contracts that ensure the correctness of transactions data before it is stored in the blockchain. This solution allows enterprises to experience the possibilities of blockchain without needing to know all sorts of technical details. Ultimately, this solution allows the reuse of Ontological models and guarantees the correct implementation of Smart Contracts.

The intersection between Smart Contracts and Ontology, is seen as a great opportunity since an ontology-based Blockchain will have an enhanced interpretability, when compared with a more traditional way of development and management [29].

This dissertation sets out to address the hypothesis of using Blockchain Smart Contracts to implement DEMO *Action Models*. DEMO methodology presents a system to capture the essence and model processes. Blockchain, on the other hand, presents a new technology that can be used for the implementation of processes.

There are already tools that resort to MDE to facilitate the development of Blockchain applications in the space of BPs and Blockchain applications. Mendling et al. [14] developed an open-source BPMS that runs entirely on the Blockchain, called Caterpillar [15]. Another example of that is Lorikeet [21] an MDE tool that can automatically generate Smart Contracts from BP models and/or registry data schema. These works prove the feasibility of executing BPs on top of Blockchains by taking process models as a starting point.

One of the most common graphic process modeling notations used is BPMN. BPMN focuses on

modeling the articulation between activities, resources, flows, gateways, events, messages, and data objects that occur in a BP. However, BPMN doesn't adequately support the articulation of business rules that would be essential for the automatic generation of Smart Contracts. Process modelers usually have to use workarounds, usually by also using additional tools, to include business rules in their processes [63]. In addition, Fickinger and Recker [76], concluded that BPMN has a level of 51.3% of overlapping language concepts and lacks state concept to ensure a more sound semantics.

DEMO, on the other hand, has been widely accepted in both scientific research and practical appliance [3, 26]. DEMO ψ-theory specifies semantic meaning to the business transactions. The business transactions dynamic includes the actors, the communications, the productions, and all its dependencies. While BPMN does not describe any semantic for the business model, it only provides a set of constructs that could be combined accordingly with a specification.

To address the problem set out, this work has presented two artifacts which consist of one conceptual mapping, between DEMO concepts and Solidity concepts, and a base contract, from which all other transaction should inherit. These artifacts aim to remove ambiguities and wrong implementations of the DEMO standard transaction pattern. These two artifacts allow for the development to only focus on the business logic implementation. By using Blockchain technology, it ensures that something that happens cannot be deleted once recorded. Moreover, using Smart Contracts ensures that a given routine is executed every time a transaction, of a given type, is recorded. This alternative method ensures that important business rules are always followed, in addition to facilitating asset control.

The prototype of a possible ontological builder was also presented, although it is not identified as one of the artefacts resulting from the work presented here. By computing a high-level formal model as DEMO *Action Model* and generating Smart Contracts from it, one can create smart contracts that ensure the correctness of transactions data before it is stored in the blockchain.

To develop this thesis, three DSRM iterations were necessary. Regarding the mapping between the DEMO concepts and Solidity concepts, DEMO ensures the syntactic validation, while the two Use Case ensure the semantic validation. As for the base contract, from which all other transactions should inherit, it instantiates the mapping, and its validation is done using the EVM compiler and Solidity language.

Two distinct Use Cases were studied. The first, presented in Chapter 5, refers to assets that are materialized outside the blockchain. In this case, the asset (car) studied, is realized outside the blockchain, but its control is done on the blockchain. The second, presented in Chapter 6, refers to assets that are materialized within the blockchain. In this case, the asset (vote) studied, is realized within the blockchain, and its control is done on the blockchain.

The evaluation has shown first that both the mapping between DEMO concepts and Solidity concept and the base contract are abstract, meaning that are generic to apply to different inter-organizational processes. Despite the existence of some works that combine DEMO with Blockchain [40, 53, 54, 89],

both artifacts are original. Both artifacts ease the development of blockchain applications and properly explains how the concepts of DEMO are mapped with the concepts of a public blockchain.

Two different assessments were made, performance, and functional arguments for the control of assets. The first is more oriented towards quantitative metrics, such as costs and latency. And the second is more oriented towards less measurable and more functional metrics, but equally important. The second mainly studies the solution in terms of its usability for communication between parties, and the re-usability of artifacts to structure and organize knowledge. With this work it was then possible to answer the research questions asked in Section 1.3.

Finally, the work of this thesis was communicated in paper format to ICEIS [1], KEOD [2], and MODELSWARD [3] to obtain a scientific evaluation and feedback.

8.2 Limitations and Future Work

This dissertation proposes a mapping between DEMO concepts and Solidity concepts through two Use Case. Although the current solution already proved valuable to address this problem, it can be further improved. In this section, a description of the limitations found and suggestions for future research are presented.

The first limitation encountered relates to the massive use of blockchain technology. There is currently no public blockchain capable of supporting millions of transactions in a cost and time-efficient way. According to the Gartner [87], production-ready blockchain technology will be available by 2030. By then, the transaction fees should be lower, so significant cost savings can be realized.

A limitation that can be easily overcome is the fact that this was the work that focused a lot on the Solidity language. This limitation can be overcome by the application of an Interpreter that reads a source code of an interpreted programming language, in this case Solidity, and converts it into executable code in another language.

As future work driven by the present work, the vehement next step would be to try to optimize this implementation. The first situation to be studied are transactions that encloses other transactions. This work believes there are three possible solutions. The presented in this work, but not optimal, the sub-transaction can be implemented as another contract. The enclosing contract than stores the address of the sub-contract. The second solution, believes there must not always be a need to create separate contract for sub-transactions, the sub-transaction can be implemented inside the main contract. This can be convenient if only partial execution on Blockchain for the sub-transactions doesn't raise an issue. Finally, the last option is that the sub-transactions are not handled at all and leave this outside

[1] http://www.iceis.org/
[2] http://www.keod.ic3k.org/
[3] http://www.modelsward.org/

of Blockchain. The choice of which option to choose seems at all related to the situation in question. Moreover, it might not be possible to implement full business logic and there is no need to run the exact same transaction execution multiplied on thousands of computers.

Execution costs on Blockchain vary by platform, which brings more responsibility when deciding which transactions or transaction steps to implement on- and off-chain. Will have to be studied which transactions or transaction steps would bring benefits of being fully or partially executed on Blockchain. For instance, if a transaction states that once a certain amount of money is paid an asset will be transferred. When implementing this transaction in Blockchain, 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 Blockchain technology brings to the table.

Comparisons between executing DEMO transactions in public and private Blockchains also seems fit. As well as, compare different Blockchain implementations such as Hyperledger Fabric and Ubiq and their ability to execute DEMO standard transaction pattern.

Another interesting path for future research would be the integration of Oracles, and its ontological implications. For instance, for the Use Case presented at Chapter 6 an oracle could substitute the *Election Controlling* transaction, as this transaction can be replaced by a timer that activates every five years.

At last, although the conceptualization and proposed a validation by means of a practical examples, a large-scale application would be an essential next step to strengthen the validation and would allow discovering some potentially remaining weakness. Use Cases and usability studies to improve the proposed method.

Bibliography

[1] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[2] B. Hofreiter, C. Huemer, and W. Klas, "ebxml: Status, research issues, and obstacles," in *Proceedings Twelfth International Workshop on Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems RIDE-2EC 2002*. IEEE, 2002, pp. 7–16.

[3] H. B. F. DIETZ, JAN L. G. MULDER, *ENTERPRISE ONTOLOGY: a human-centric approach to understanding the essence of organisation*. SPRINGER NATURE, 2020.

[4] "Why model-driven engineering fits the needs for blockchain application development." [Online]. Available: https://blockchain.ieee.org/technicalbriefs/september-2018/why-model-driven-engineering-fits-the-needs-for-blockchain-application-development

[5] M. Müller, N. Ostern, and M. Rosemann, "Silver bullet for all trust issues? blockchain-based trust patterns for collaborative business processes," *Blockchain for Trust-aware Business Processes - Preprint*, 07 2020.

[6] J. Dietz, "Demo-4 specification language," *Enterprise Engineering Institute*, 2019.

[7] M. Op't Land, E. Proper, M. Waage, J. Cloo, and C. Steghuis, *Enterprise architecture: creating value by informed governance*. Springer Science & Business Media, 2008.

[8] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer Berlin, 2019.

[9] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[10] S. Lund, J. Manyika, J. Woetzel, J. Bughin, M. Krishnan, J. Seong, and M. Muir, "Globalization in transition: The future of trade and value chains," Oct

2019. [Online]. Available: https://www.mckinsey.com/featured-insights/innovation-and-growth/globalization-in-transition-the-future-of-trade-and-value-chains#

[11] K. C. Laudon and C. G. Traver, *E-commerce: business. technology. society*. Pearson, 2020.

[12] M. van Wingerde and H. Weigand, "An ontological analysis of artifact-centric business processes managed by smart contracts," in *2020 IEEE 22nd Conference on Business Informatics (CBI)*, vol. 1. IEEE, 2020, pp. 231–240.

[13] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997. [Online]. Available: https://firstmonday.org/ojs/index.php/fm/article/view/548

[14] J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar *et al.*, "Blockchains for business process management-challenges and opportunities," *ACM Transactions on Management Information Systems (TMIS)*, vol. 9, no. 1, pp. 1–16, 2018.

[15] O. López-Pintado, L. García-Bañuelos, M. Dumas, and I. Weber, "Caterpillar: A blockchain-based business process management system." in *BPM (Demos)*, 2017.

[16] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, "Untrusted business process monitoring and execution using blockchain," in *Business Process Management*, M. La Rosa, P. Loos, and O. Pastor, Eds. Cham: Springer International Publishing, 2016, pp. 329–347.

[17] Gartner, "The reality of blockchain," https://www.gartner.com/smarterwithgartner/the-reality-of-blockchain/, 2019, accessed: 2020-08-10.

[18] ——, "Gartner survey reveals the scarcity of current blockchain deployments," https://www.gartner.com/en/newsroom/press-releases/2018-05-03-gartner-survey-reveals-the-scarcity-of-current-blockchain, 2018, accessed: 2020-08-10.

[19] A. Norta, "Designing a smart-contract application layer for transacting decentralized autonomous organizations," in *Advances in Computing and Data Sciences*, M. Singh, P. Gupta, V. Tyagi, A. Sharma, T. Ören, and W. Grosky, Eds. Singapore: Springer Singapore, 2017, pp. 595–604.

[20] X. Xu, I. Weber, and M. Staples, "Model-driven engineering for blockchain applications," in *Architecture for Blockchain Applications*. Springer, 2019, pp. 149–172.

[21] A. B. Tran, Q. Lu, and I. Weber, "Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management." in *BPM (Dissertation/Demos/Industry)*, 2018, pp. 56–60.

[22] M. Snoeck, “Enterprise information systems engineering,” *The MERODE Approach*, 2014.

[23] J. A. P. Hoogervorst, *Enterprise governance & architectuur: corporate, IT en enterprise governance in samenhangend perspectief*. Academic Service, 2007.

[24] J. L. Dietz, “Enterprise ontology-understanding the essence of organizational operation,” in *Enterprise Information Systems VII*. Springer, 2007, pp. 19–30.

[25] ——, *What is Enterprise Ontology?* Springer, 2006.

[26] M. Andrade, D. Aveiro, and D. Pinto, “Demo based dynamic information system modeller and executer,” *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, 2018.

[27] J. Dietz and J. Hoogervorst, “Enterprise ontology and enterprise architecture–how to let them evolve into effective complementary notions,” *GEAO Journal of Enterprise Architecture*, vol. 2, no. 1, pp. 121–149, 2007.

[28] “Not-so-clever contracts.” [Online]. Available: https://www.economist.com/business/2016/07/28/not-so-clever-contracts?fsrc=scn/tw/te/pe/ed/notsoclevercontracts

[29] H. Kim and M. Laskowski, “Towards an ontology-driven blockchain design for supply chain provenance,” in *Wiley Online Library*, 08 2016.

[30] S. Yongchareon, J. Yu, X. Zhao *et al.*, “A view framework for modeling and change validation of artifact-centric inter-organizational business processes,” *Information systems*, vol. 47, pp. 51–81, 2015.

[31] A. Nigam and N. S. Caswell, “Business artifacts: An approach to operational specification,” *IBM Systems Journal*, vol. 42, no. 3, pp. 428–445, 2003.

[32] R. Hull, “Artifact-centric business process models: Brief survey of research results and challenges,” in *OTM Confederated International Conferences” On the Move to Meaningful Internet Systems”*. Springer, 2008, pp. 1152–1163.

[33] B. K. Mohanta, S. S. Panda, and D. Jena, “An overview of smart contract and use cases in blockchain technology,” in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, July 2018, pp. 1–4.

[34] K. Zīle and R. Strazdiņa, “Blockchain use cases and their feasibility,” *Applied Computer Systems*, vol. 23, no. 1, pp. 12 – 20, 2018. [Online]. Available: https://content.sciendo.com/view/journals/acss/23/1/article-p12.xml

[35] J. Mattila, “The blockchain phenomenon,” *Berkeley Roundtable of the International Economy*, 2016.

[36] S. Haber and W. S. Stornetta, “How to time-stamp a digital document,” *J. Cryptol.*, vol. 3, no. 2, pp. 99–111, Jan. 1991. [Online]. Available: http://dx.doi.org/10.1007/BF00196791

[37] M. Jakobsson and A. Juels, *Proofs of Work and Bread Pudding Protocols(Extended Abstract)*. Boston, MA: Springer US, 1999, pp. 258–272. [Online]. Available: https://doi.org/10.1007/978-0-387-35568-9_18

[38] A. Bahga and V. Madisetti, “Blockchain platform for industrial internet of things,” *Journal of Software Engineering and Applications*, vol. 09, pp. 533–546, 01 2016.

[39] M. Bartoletti and L. Pompianu, “An empirical analysis of smart contracts: Platforms, applications, and design patterns,” *Lecture Notes in Computer Science*, 03 2017.

[40] B. Hornáčková, M. Skotnica, and R. Pergl, “Exploring a role of blockchain smart contracts in enterprise engineering,” in *Advances in Enterprise Engineering XII*, D. Aveiro, G. Guizzardi, S. Guerreiro, and W. Guédria, Eds. Cham: Springer International Publishing, 2019, pp. 113–127.

[41] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, and A. Ponomarev, “Caterpillar: A business process execution engine on the ethereum blockchain,” 2019.

[42] C. Dannen, *Introducing Ethereum and Solidity: foundations of cryptocurrency and blockchain programming for beginners*. Apress, 2017.

[43] M. Swan, *Blockchain : blueprint for a new economy*. Sebastopol, Calif.: O’Reilly Media, 2015. [Online]. Available: http://shop.oreilly.com/product/0636920037040.do

[44] D. C. Schmidt, “Guest editor’s introduction: Model-driven engineering,” *Computer*, vol. 39, no. 2, pp. 25–31, Feb 2006.

[45] B. Scott, J. Loonam, and V. Kumar, “Exploring the rise of blockchain technology: Towards distributed collaborative organizations,” *Strategic Change*, vol. 26, no. 5, pp. 423–428, 2017.

[46] J. L. G. Dietz, *Enterprise ontology: theory and methodology*. Springer, 2011.

[47] S. M. Cohen, “Aristotle’s metaphysics,” in *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2016.

[48] N. Guarino, D. Oberle, and S. Staab, “What is an ontology?” in *Handbook on Ontologies*, 05 2009, pp. 1–17.

[49] M. Verdonck, F. Gailly, S. de Cesare, and G. Poels, “Ontology-driven conceptual modeling: A systematic literature mapping and review,” *Applied Ontology*, vol. 10, pp. 197–227, 12 2015.

[50] B. A. Tama, B. J. Kweka, Y. Park, and K. Rhee, “A critical review of blockchain and its current applications,” in *2017 International Conference on Electrical Engineering and Computer Science (ICECOS)*, 2017, pp. 109–113.

[51] X. Xu, C. Pautasso, L. Zhu, Q. Lu, and I. Weber, “A pattern collection for blockchain-based applications,” *Proceedings of the 23rd European Conference on Pattern Languages of Programs*, 2018.

[52] F. Idelberger, G. Governatori, R. Riveret, and G. Sartor, “Evaluation of logic-based smart contracts for blockchain systems,” in *RuleML*, 2016.

[53] J. Kruijff and H. Weigand, “Understanding the blockchain using enterprise ontology,” 06 2017.

[54] J. de Kruijff and H. Weigand, “Understanding the blockchain using enterprise ontology,” in *Advanced Information Systems Engineering*, E. Dubois and K. Pohl, Eds. Springer International Publishing, 2017, pp. 29–43.

[55] M. Aparício., S. Guerreiro., and P. Sousa., “Towards an automated demo action model implementation using blockchain smart contracts,” in *Proceedings of the 22nd International Conference on Enterprise Information Systems - Volume 2: ICEIS,*, INSTICC. SciTePress, 2020, pp. 762–769.

[56] C. K. Frantz and M. Nowostawski, “From institutions to code: Towards automated generation of smart contracts,” in *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, Sep. 2016, pp. 210–215.

[57] S. E. S. Crawford and E. Ostrom, “A grammar of institutions,” *American Political Science Review*, vol. 89, no. 3, p. 582–600, 1995.

[58] G. Boella, E. Elkind, B. T. R. Savarimuthu, F. Dignum, and M. K. Purvis, “Prima 2013: Principles and practice of multi-agent systems,” in *Lecture Notes in Computer Science*, 2013.

[59] A. Mavridou and A. Laszka, “Designing secure ethereum smart contracts: A finite state machine based approach,” in *Financial Cryptography and Data Security*, S. Meiklejohn and K. Sako, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018, pp. 523–540.

[60] J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar, A. Gal, L. García-Bañuelos, G. Governatori, R. Hull, M. L. Rosa, H. Leopold, F. Leymann, J. Recker, M. Reichert, H. A. Reijers, S. Rinderle-Ma, A. Solti, M. Rosemann, S. Schulte, M. P. Singh, T. Slaats, M. Staples, B. Weber, M. Weidlich, M. Weske, X. Xu, and L. Zhu, “Blockchains for business process management - challenges and opportunities,” *ACM Trans. Manage. Inf. Syst.*, vol. 9, no. 1, pp. 4:1–4:16, Feb. 2018. [Online]. Available: http://doi.acm.org/10.1145/3183367

[61] L. García-Bañuelos, A. Ponomarev, M. Dumas, and I. Weber, "Optimized execution of business processes on blockchain," in *BPM*, 09 2017.

[62] W. Hu, Z. Fan, and Y. Gao, "Research on smart contract optimization method on blockchain," *IT Professional*, vol. 21, no. 5, pp. 33–38, Sep. 2019.

[63] M. Aparício., S. Guerreiro., and P. Sousa., "Automated demo action model implementation using blockchain smart contracts," in *Proceedings of the 12th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 2: KEOD,*, INSTICC. SciTePress, 2020, pp. 283–290.

[64] O. Mráz, P. Náplava, R. Pergl, and M. Skotnica, "Converting demo psi transaction pattern into bpmn: A complete method," in *Advances in Enterprise Engineering XI*, D. Aveiro, R. Pergl, G. Guizzardi, J. P. Almeida, R. Magalhães, and H. Lekkerkerk, Eds. Cham: Springer International Publishing, 2017, pp. 85–98.

[65] J. A. Garcia-Garcia, N. Sánchez-Gómez, D. Lizcano, M. J. Escalona, and T. Wojdyński, "Using blockchain to improve collaborative business process management: Systematic literature review," *IEEE Access*, vol. 8, pp. 142 312–142 336, 2020.

[66] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele Univ.*, vol. 33, 08 2004.

[67] V. Amaral de Sousa, C. Burnay, and M. Snoeck, "B-merode: A model-driven engineering and artifact-centric approach to generate blockchain-based information systems," in *Advanced Information Systems Engineering*, S. Dustdar, E. Yu, C. Salinesi, D. Rieu, and V. Pant, Eds. Cham: Springer International Publishing, 2020, pp. 117–133.

[68] M. Snoeck, C. Michiels, and G. Dedene, "Consistency by construction: The case of merode," in *Conceptual Modeling for Novel Application Domains*, M. A. Jeusfeld and Ó. Pastor, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 105–117.

[69] R. Hull, "Artifact-centric business process models: Brief survey of research results and challenges," in *On the Move to Meaningful Internet Systems: OTM 2008*, R. Meersman and Z. Tari, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 1152–1163.

[70] D. M. Schunselaar, J. Gulden, H. van der Schuur, and H. A. Reijers, "A systematic evaluation of enterprise modelling approaches on their applicability to automatically generate erp software," in *2016 IEEE 18th Conference on Business Informatics (CBI)*, vol. 1. IEEE, 2016, pp. 290–299.

[71] H. Weigand, "The e3value ontology for value networks-current state and future directions," *Journal of Information Systems*, vol. 30, 02 2016.

[72] F. Hunka and J. Zacek, “Detailed analysis of rea ontology,” in *Advances in Enterprise Engineering VIII*, D. Aveiro, J. Tribolet, and D. Gouveia, Eds. Cham: Springer International Publishing, 2014, pp. 61–75.

[73] K. Markov, “Data independence in the multi-dimensional numbered information spaces,” *International Journal*, vol. 2, 01 2008.

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

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

[76] F. Tobias and R. Jan, “Construct redundancy in process modelling grammars: Improving the explanatory power of ontological analysis,” *ECIS 2013 - Proceedings of the 21st European Conference on Information Systems*, 01 2013.

[77] C. Dannen, *Introducing Ethereum and Solidity: foundations of cryptocurrency and blockchain programming for beginners*. Apress, 2017.

[78] M. Wöhrer and U. Zdun, “Design patterns for smart contracts in the ethereum ecosystem,” in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018, pp. 1513–1520.

[79] P. Ceravolo, C. Fugazza, and M. Leida, “Modeling semantics of business rules,” in *2007 Inaugural IEEE-IES Digital EcoSystems and Technologies Conference*, 2007, pp. 171–176.

[80] K. Taveter and G. Wagner, “Agent-oriented enterprise modeling based on business rules,” in *Conceptual Modeling — ER 2001*, H. S.Kunii, S. Jajodia, and A. Sølvberg, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 527–540.

[81] O. Holschke, P. Gelpke, P. Offermann, and C. Schröpfer, “Business process improvement by applying reference process models in soa - a scenario-based analysis,” in *Multikonferenz Wirtschaftsinformatik*, 2008.

[82] P. W. Hare, *Making diplomacy work: intelligent innovation for the modern world*. CQ Press, 2015.

[83] M. Patil, V. Pimplodkar, A. R. Zade, V. Vibhute, and R. Ghadge, “A survey on voting system techniques,” *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 1, pp. 114–117, 2013.

[84] K. Curran, “E-voting on the blockchain,” *The Journal of the British Blockchain Association*, vol. 1, no. 2, p. 4451, 2018.

[85] S. Voshmgir, *Token economy: How blockchains and smart contracts revolutionize the economy*. BlockchainHub, 2019.

[86] A. Das, W. Wu, and D. L. McGuinness, “Industrial strength ontology management,” in *Proceedings of the First International Conference on Semantic Web Working*. CEUR-WS. org, 2001, pp. 17–37.

[87] Gartner, “The reality of blockchain,” 2019. [Online]. Available: https://www.gartner.com/smarterwithgartner/the-reality-of-blockchain/

[88] T. Bürger and E. Simperl, “Measuring the benefits of ontologies,” in *On the Move to Meaningful Internet Systems: OTM 2008 Workshops*, R. Meersman, Z. Tari, and P. Herrero, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 584–594.

[89] D. Silva, S. Guerreiro, and P. Sousa, “Decentralized enforcement of business process control using blockchain,” *Advances in Enterprise Engineering XII Lecture Notes in Business Information Processing*, p. 69–87, 2018.

A

Related Work Tables

An overview of all patterns concerning the taxonomy can be seen in Figure A.1.

Pattern Name (Section)	Trust-enhancing Method	Uncertainty Root	Trust Concern	Process Component	Limitations
Hash Storage (4.1)	reduce uncertainty	data	integrity	data object	data processing outside scope
Transparent Event Log (4.2)	reduce uncertainty	organization	non-repudiation	event	initial event submission must be ensured
Blockchain BP Engine (4.3)	reduce uncertainty	resource, activity, organization	integrity	gateway, sequence flow, message flow	time-based logic
Smart Contract Activities (4.4)	reduce uncertainty	activity	integrity, availability	activity	oracle, privacy
Blockchain-based Reputation System (4.5)	build confidence	organization	integrity	any	reputation system attacks
Decentralization (Section 4.6)	build confidence	organization	integrity	any	incentives

Table 1. Comparison of blockchain-based trust patterns

Figure A.1: Architecture of Lorikeet. Table retrieved from [5].

Figure A.2 shows an abstract indication of the influence of the blockchain technology on the uncertainties in a BP: Hash Storage; Transparent Event Log; Blockchain BP Engine; and Smart Contract Activities. The letters indicate which blockchain-based trust pattern can be used to enhance which trust concern.

	Data	Activity	Event	Gateway	Sequence Flow	Message Flow
Integrity	HS	SCA	TEL	BPE, SCA	BPE	BPE
Confidentiality						
Availability		SCA	SCA	SCA	SCA	SCA
Non-repudiation		TEL	TEL	TEL	TEL	TEL
Performance						
Resilience						

Table 2. Impact potential of blockchain to mitigate process related uncertainties: HS - Hash Storage, TEL - Transparent Event Log, BPE - Blockchain Business Process Engine, SCA - Smart Contract Activities

Figure A.2: Architecture of Lorikeet. Table retrieved from [5].