Utrecht University

Master Thesis

Business Informatics

A Model-Driven Approach to Smart Contract Development

Deloitte.

First supervisor:

dr. F. Dalpiaz

Second supervisor:

dr. M.R. Spruit

First external supervisor:

J. van Dalen

Second external supervisor:

A. Beentjes

Author:

Kees Boogaard

K.Boogaard@students.uu.nl

June, 2018

Abstract

Blockchain technology has provided a platform for the decentralized execution of smart contracts. A smart contract is an agreement that is automatically executed when certain conditions have been met. The immutability, decentral nature, and consensus mechanisms that are characteristic to blockchain technology make the smart contract and its development cycle a new field of study in software engineering. A novel economic and defensive thinking is needed to develop workable, secure smart contracts. Motivated by the need for a novel approach to development, this thesis proposes a model-driven approach to smart contract development.

Model-Driven Engineering (MDE) is an approach to information system development in which models and model technologies are applied to raise the level of abstraction at which developers create and evolve software, with the goal of both simplifying and formalizing the various activities and tasks that comprise the Software Development Life Cycle (SDLC). Model-Driven Architecture (MDA) is a framework for this approach. This thesis aims to apply this framework to create a method which describes the development phase from domain knowledge to smart contract foundation.

The creation of a method has two main aims, namely (i) to bridge the semantic gap between domain knowledge and smart contract by lowering the threshold for domain experts, and (ii) support developers in creating less vulnerable smart contracts that accurately represent the problem domain. This is done by constructing a model-driven method based on existing research that applies MDE to smart contract development. A literature study into this field yields the requirements and techniques for the method, which is consequently constructed based on these requirements and techniques.

The method is evaluated in twofold. First, the value is assessed through a case study, which shows that the developer benefits from a structured approach and the reduction of manual programming. Second, by an experiment which shows that people are better able to comprehend and communicate about models containing functional aspects of the smart contract if a computational independent model is included. By doing so it fulfills the aim of lowering the threshold for domain experts to participate in the smart contract development cycle.

Preface

The process of creating this master thesis started out roughly a year ago when I was exploring possible topics and fields of interest. After a period of exploration the decision to do "something with blockchain" marked the start of this research. The possibilities to innovate and the wide variety of applications were what drew me into this field of study. Around this time, Deloitte was kind enough to offer me a place to conduct this research. I aimed to explore the possible impact of blockchain technology within risk advisory, with a focus on IT assurance. It turned out that this topic had already been explored, so I shifted my focus from the possible influence of blockchain technology itself, to an exploration of the actual application of the technology. Since then, the topic has shifted from information security within smart contracts, to the application of analysis tools, to the final subject: the smart contract development process.

This master thesis is the result of nine months of research at Deloitte, during which I have learned more than I could have imagined I would. Lessons learned both academic, as well as on the fantastic Deloitte working culture. From this spot, I would like to express my gratitude to Jordi van Dalen, Arne Beentjes, and all Deloitte Risk Advisory colleagues for their insights, feedback, and overall contributions to this thesis. I would also like to thank Fabiano Dalpiaz for his feedback and continuous guidance throughout this research.

To conclude, I hope this master thesis will present valuable insights to you as a reader. Enjoy reading!

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ABM	Agent-Based Modeling
BPMN	Business Process Model and Notation
CIM	Computational Independent Model
DAO	Decentralized Autonomous Organization
EMF	Eclipse Modeling Framework
EVM	Ethereum Virtual Machine
FSM	Finite State Machine
GQM	Goal-Question-Metric
JMI	Java Metadata Interface
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDE	Model-Driven Engineering
MOF	MetaObject Facility
OMG	Object Management Group
PDD	Process-Deliverable Diagram
PIM	Platform Independent Model
PoW	Proof-of-Work
SDLC	Software Development Life Cycle
SME	Situational Method Engineering
UML	Unified Modeling Language

1. Introduction

In the ever-evolving world of computing and technology there are often technologies which are called revolutionary or ground-breaking. In 2008, the pseudonymous Satoshi Nakamoto presented the paper "Bitcoin, a peer-to-peer electronic cash system" [1]. This paper proposed an electronic currency which was not governed by a centralized authority. A year later the Bitcoin blockchain was released and the technology which made this decentralized approach possible was coined blockchain technology. Almost a decade later the net worth of Bitcoin has skyrocketed, other applications aimed at electronic cash called cryptocurrencies are numerous, and the Blockchain technology is often referred to as the most disruptive technological innovation since the internet. This disruption claim is based on the large variety of use cases in which Blockchain can be applied. These use cases are for instance financial systems [2], the internet of things [3], and supply chains [4]. Essentially, a blockchain is an append-only data structure maintained by the nodes of a peer-to-peer network. It is a decentralized, immutable, and near-real-time way to store data [5]. These properties are used in cryptocurrencies by using the blockchain as a public ledger which records all transactions.

The novel way of data storage made possible by blockchain technology has sprouted numerous other blockchain platforms. Based on the market capitalization, Bitcoin is the largest platform [6]. The second largest is Ethereum, which is the main focus of this thesis. While Bitcoin and platforms alike are mainly focused on keeping a ledger of who owns what in order to create a digital currency, Ethereum is a general purpose blockchain. This means that the blockchain can understand a general-purpose programming language and serves as a distributed computing platform. This allows developers to create applications for the Ethereum blockchain instead of having to build an entirely new blockchain platform for each use case. Next to the possibility of creating applications, the Ethereum platform also supports peer-to-peer currency transfer like Bitcoin does.

The peer-to-peer currency transfer has a major downside: their highly volatile value. As the popularity of blockchain platforms gained momentum, so did the speculation mania on the currencies and as of this moment it is highly uncertain if

people are buying cryptocurrency in order to use it as a currency or as a speculative asset [7]. Important instigators of this volatility are the regulators, so governments who are reacting to this new form of currency [8]. Rumors about possible regulations can lead to a rise or drop of tens of percent's an hour. This is especially difficult for a platform like Ethereum, in which the computations are paid for in currency and the program commands are activated through transactions. Speculation on Ether, the currency of Ethereum, could potentially jam the entire network, resulting in contracts not executing. Ideally, the cryptocurrencies are not used as speculative assets so that the price in US$ has no effect on the blockchain platforms.

The key component that differentiates Ethereum from most other blockchain platforms is that it is able to understand a general purpose language. This allows developers to create programs that run on the blockchain. These programs are called smart contracts. The term smart contract was introduced by Nick Szabo in 1997, who describes a smart contract as "*a set of promises, specified in digital form, including protocols within which the parties perform on these promises*" [9]. The idea is described as moving contractual clauses into hardware and software in such a way that breaking the contract becomes expensive. Szabo did not have a specific system for implementation in mind, but some trust in a third party was assumed. The idea of smart contracts has rapidly regained momentum with the emergence of blockchain technology, which solves the problem of trust through a consensus protocol.

Smart contracts and a blockchain platform that can execute them has a variety of possible use cases. However, as of this moment the biggest implementations of smart contracts have been plagued by problems. For instance, the best-known example in Ethereum is the Decentralized Autonomous Organization (DAO) [10]. This smart contract served as a crowdfunding application and the participants were able to buy, give away, or retrieve tokens. By exploiting a vulnerability in the code, a hacker was able to drain 3.6 million ether from the contract, which was worth around 50 million dollars at the time. This shows that not accounting for possible security threats can be costly in terms of money. However, in the case of the DAO it was not only in costly in terms of money, but also the reputation and the belief in the Ethereum platform was damaged. For a platform that is built upon trust in the peers and the technology, it goes without saying that this is especially detrimental.

1.1 Problem Statement

Blockchain technology and smart contracts have experienced a steady increase of attention from academia and the industry [11][12]. Out of all the blockchain platforms, Ethereum is the first big outing of a decentralized computing system in which the nodes execute smart contracts. Before the smart contracts can be appended to the blockchain, they have to be developed first. The development of smart contracts is different from classical software development in a number of ways. Two of the biggest differences and adjacent challenges of smart contracts lay in the immutability and the availability properties of blockchain technology. The immutability of blockchain means that once a smart contract is appended to the blockchain, it cannot be modified. As of this moment, there is no way to patch a buggy smart contract, regardless of its popularity or how much currency it holds, without reversing the blockchain or relaunching the smart contract. Next to the pressure of getting it absolutely right the first attempt, the virtual currencies have real value. This means that if you load money or currency into a buggy smart contract, you may lose it [13]. The value of the currency of Ethereum, Ether, has increased tremendously over the course of Ethereum's existence and as a result, some smart contracts are now worth millions of dollars [14].

The combination of the worth of the smart contracts with the availability property of blockchain, meaning that all nodes have access to it, are the reason that smart contracts and the Ethereum platform are a continuous target for attackers. Numerous security vulnerabilities have been uncovered. The openly accessible nature and the large possibility of counterparties attempting to execute a contract maliciously call for a unique kind of defensive thinking in the development of smart contracts [15].

In 2016, a symbolic execution analysis tool was developed by Luu et al. [16], which analyzed 19,366 smart contracts deployed on Ethereum. Their results show that 45% of these were vulnerable to at least one security vulnerabilities. Since 2016, the amount of smart contracts deployed has grown exponentially and although there are static analysis tools available, it is expected that a lot of these contracts still have vulnerabilities. Although a large number of these vulnerabilities will not be exploited, many of them enable cybercriminals to steal digital assets.

Smart contracts differ from normal contracts in the sense that they are self-executing and are interpreted by computers, not by intermediaries. In this thesis the

terms paper contract, normal contract, and classical contract are used interchangeably to denote non-smart contracts. For non-programmers, it is difficult to express contracts into code, and vice-versa it is difficult for programmers to fully grasp the requirements of a contract in the sense that all the domain concepts should be transferred into a smart contract correctly while accounting for security vulnerabilities. In a study on a smart contract programming class [15], researchers found that a unique economic thinking was needed that a traditional programmer may lack. Logical errors can lead to currency leakage, and its transparency and availability leads to security vulnerabilities. The study also shows that the learning curve for the development of smart contracts is steep and that there are a lot of common pitfalls for inexperienced programmers. Luu et al. state that a lot of the vulnerabilities in practice are caused by a semantic gap between the assumption contract writers make about the underlying execution semantics and the actual semantics of the smart contract system [16], i.e. the code does not work the way the writer thinks it is going to work.

The problems in smart contract can be viewed from two different viewpoints, namely from the developers viewpoint and the viewpoint from the person experienced in creating contracts, the domain expert. The developers have difficulties transitioning to the novel approach of smart contract development, and the domain experts do not possess the technological expertise to transition paper contracts into smart contracts.

To summarize, smart contracts are hard to develop and many of the deployed contracts have security vulnerabilities. A lot of these issues seem to stem from a lack of understanding of the programming language Solidity, and by a general lack of programming knowledge in the smart contract domain. The biggest advantage of smart contracts is that they, in comparison with traditional financial contracts, carry the promise of low legal and transactions costs, and can lower the bar of entry for users. However, through the difficulty of smart contract development, this bar of entry for users remains at a high level. Furthermore, the value of the blockchain currencies is highly volatile which can lead to new platforms facilitating smart contracts in different ways. A lack of formalization in the development process of smart contract contributes to these problems, so the aim of this research is to formalize

the smart contract development process in order to (i) bridge the semantic gap between domain knowledge and smart contract by lowering the threshold for domain experts, and (ii) support developers in creating less vulnerable smart contracts that accurately represent the problem domain.

1.2 Research Focus

There are numerous problems surrounding the relatively novel field of smart contract development and these problems are fairly diverse in nature. There is no easy way to pinpoint at which phase of the development the problems are nested. As stated in the problem statement, the problems range from the transformation of the domain knowledge to the developers coding vulnerable smart contracts.

As we take a look into earlier work in software development, a lot of the same problems have arose and numerous solutions have been proposed [17]. Solutions such as having different approaches to a software engineering project, like the waterfall model, the incremental model, or the evolutionary model. Furthermore, there have been attempts at different process models, such as the incremental process model or concurrent process models [18]. One thing that can be taken away from these attempts to better the software engineering practice is that they aim to bring structure to the development process through formalization.

The same can be done in the smart contract development field. Currently, there are little to no specialized formalized approaches to this development field. This leads to common pitfalls parallel to those in the overarching field of software engineering. One of these pitfalls is that the domain concepts are translated to software technology concepts entirely by mental work of the software developer [19]. This often results in a misalignment of requirements and product. Model-Driven Engineering (MDE) could assist in resolving this misaligned, while simultaneously assisting the developer in creating higher quality smart contracts in which vulnerabilities are accounted for. MDE is an approach to software engineering in which models and transformation between models are used to assist in the transition between domain knowledge and software product [20].

By providing the participants in the smart contract development process with a structured approach, for instance a method, the problems addressed in the problem

statement can possibly be alleviated. Therefore, the main aim of this research is to address the problems surrounding smart contracts by providing a structured method for its development which applies the concept of MDE to the field of smart contract development.

1.3 Contributions

The research presented in this thesis adds value to the knowledge base in a number of ways. First, it presents a holistic definition of the concepts blockchain and smart contracts, and an overview of the current state of the research into the application of MDE to smart contracts. Research into blockchain and smart contracts is partly nested in academia, but for a large part it is done in an online open-source setting in which willing participants contribute and build upon the work of others without the need for extensive documentation. For this reason, it is useful to have a scientifically written overview on the current state of smart contract development.

Second, the smart contract development community is helped by insights provided in this research. Smart contract developers should aim to create high quality smart contracts as efficient as possible, so a development method supporting this aim can contribute to both the quality, as well as the lowering development time. The application of smart contracts gains momentum and it is expedient for the smart contract to have as little vulnerabilities as possible, while still representing the domain it should reflect. The method presented in this research contributes to these goals.

Third, the method facilitates a communication tool for domain experts and developers to communicate about what domain concepts should be transformed into technology concepts in a smart contract. This can be a step toward a solution for the misalignment between domain and smart contract.

1.4 Scope

Blockchain and smart contracts are broad topics in which a lot of research is still to be done. This research aims solely on the development phase of the smart contract in order to prevent a generic research project in which all aspects are treated in a shallow way. This does mean that there is little mentioned in this research on the quality of blockchains and possible innovations. Ethereum is taken as the blockchain of interest in this research, simply because it is by far the biggest distributed computing platform

as of this moment. An advantage of applying MDE is that when an improved blockchain platform surfaces, only the transformation rules need to be adjusted in order to retain the relevance of the method.

The method will be aimed at the development phase of smart contract, so not on the requirements engineering phase. This process requires a research of its own and is outside the scope of this thesis. As of this moment, it is not possible to alter smart contracts as they are launched to the blockchain, so the maintenance phase is not relevant in this context. However, with MDE, alteration of a contract is made into an easier process as the documentation gives a better overview of what should be replaced, added, or deleted.

1.5 Thesis outline

In the second chapter, the research approach is explained, containing the research questions, the research paradigm, and the literature research protocol for this thesis. Hereafter in chapter 3, the theoretical foundations of this thesis are described. Based on the theoretical foundations, chapter 4 will describe the creation of the model-driven smart contract development method. Chapter 5 will assess this method through a case study, and chapter 6 will evaluate the comprehension of the models used through an experiment. The conclusions that this thesis yields will be discussed in chapter 7, and the validity of the research conducted will be discussed in chapter 8. Lastly, future research directions are discussed in chapter 9.

2. Research Approach

Good research starts with a well-defined research approach. This chapter describes the research approach for the subject at hand. First, the research questions and sub-questions are described and elaborated upon. After this, the choice and instrumentation of the design science paradigm is explained. Lastly, this chapter describes the literature research protocol used for the collection of knowledge.

2.1 Research Questions

To achieve the goals denoted in the previous chapter, a main research question has been formulated. This main research question is aimed at adding structure to the development of smart contracts through MDE. The structured way denoted in the research question is refers to the formalization of the development process.

RQ: *How can smart contract development be supported by Model-Driven Engineering in a structured way?*

To answer this research question, a number of sub-goals need to be achieved. The first sub-goal is achieving an understanding of the concepts used in this research, namely blockchain, smart contracts, and MDE. This will be achieved by looking into prior research on the topics, by means of a literature review on the concepts. This results in the following research sub-question:

SQ1: *How can blockchain, smart contracts, and model-driven engineering be defined based on prior literature?*

With the knowledge resulting from research sub-question 1, an overview of the concepts is created. With a clear definition and understanding of these concepts, the next step in this research will be a review into the overlap of these concepts. Based on initial research, the expectation is that there already have been attempts at combining smart contract development with MDE, but not yet in a formalized structured way this thesis intends to provide. The second research sub-question is as follows:

SQ2: *What research into the application of Model-Driven Engineering to smart contract development has already been conducted?*

The insights of SQ1 and SQ2 will provide an overview of the current state of research into smart contract development and MDE. Using these insights, a method engineering process can be initiated. A method engineering approach based on using existing method fragments will be applied. This means that a method base is created, and method fragments from this method base are selected based on the requirements for the smart contract development method. In order to be able to select the most adequate fragments, the requirements for the method need to be stated. Based on the requirements and available method fragments, the method engineering process can proceed to the creation of the method. Method engineering as a concept will be explained in a later chapter, along with how it is used to create the smart contract development method. The requirements, activities, and deliverables of the method will be explored in the third research sub-question:

SQ3: *What are the requirements for the model-driven smart contract development method and what are the activities and deliverables of this method?*

The result of SQ3 will be a process deliverable diagram (PDD) that describes the method of model-driven smart contract development. After this, an evaluation of the method is necessary to investigate if the method meets the desired goals. The main goals of this research, stated in chapter 1.2, are (i) to bridge the semantic gap between domain knowledge and smart contract by lowering the threshold for domain experts, and (ii) support developers in creating less vulnerable smart contracts that accurately represent the problem domain. The first goal will be evaluated through a case study in which the method is demonstrated and the second goal will be evaluated through an experiment in which the comprehension will be tested. These evaluations are formulated through the following research sub-questions:

SQ4: : *How does the model-driven smart contract development method assist the developer in the creation of smart contracts?*

SQ5: *How does the model-driven smart contract development method influence the comprehension of smart contracts?*

The answers to the research sub-questions should provide a holistic view of the creation and evaluation of the model-driven smart contract development method,

which will provide a framework to answer the main research question. The sub-research questions will be answered by using the following approaches:

- **Sub-question 1** will be answered by doing a literature review. Smart contract development is a relatively young field, so the literature research protocol will be tailored to these circumstances. The literature research protocol can be found in chapter 2.3. Blockchain, smart contracts, and MDE will be defined separately in this sub-question and form the basis of this thesis.
- **Sub-question 2** will also be answered by doing a literature review, based on the knowledge gathered from sub-question 1. The literature that treats the interrelation between the concepts will be discussed in this sub-question. The attempts to apply MDE to smart contract development will form a method base from which the method fragments will be selected.
- **Sub-question 3** forms the basis for the method by stating the goals of the method and the available method fragments that can adhere to these goals. These goals and method fragments form the requirements for the method. Based on these requirements, a comprehensive description of the model-driven smart contract development method will be made. The knowledge from subquestion 1 and 2 will be combined to form a method that spans from the domain knowledge to the launch of the smart contract.
- **Sub question 4** will be answered by applying the method in a case study, thus demonstrating the method.
- **Sub-question 5** will be answered by describing the experiment and its outcomes. The experiment is aimed at evaluating the part of the method that is included to bridge the semantic gap between domain knowledge and smart contract. The experiment follows the goal-question-metric buildup and follows the experimental design framework.

2.2 Research Paradigm

In the Information Systems discipline, research is often characterized by either behavioural science or design science [21]. The behavioural science paradigm aims to develop theories that explain or predict the behaviour of people or organization. Design science is a paradigm that is motivated by the desire to improve the environment by the introduction of new and innovative artefacts and the processes for building these artefacts [22]. The environment denotes the problem space in which the phenomena of interest reside. In IS research, it is composed of people, organizations, and their existing or planned technologies [23]. Behaviour and design cannot be seen as two separate paradigms but have an overlap in describing theories (behaviour) and utility (design) [24]. The motivation for design science is in line with the motivation for this research, as the aim is to improve the field of smart contract development by introducing a novel artefact to this field.

Hevner has made an attempt to formalize the design science paradigm through a three cycle view [25] and a set of seven guidelines, which supports the three cycle view. The three cycle view consists of the Relevance cycle, the Rigor Cycle, and the Design Cycle. The Relevance Cycle bridges the contextual environment of the research project with the design science activities. The Rigor Cycle connects the design science activities with the knowledge base of scientific foundations, experience, and expertise that informs the research project. The central Design Cycle iterates between the core activities of building and evaluating the design artefacts and processes of the research [25]. The framework strikes a balance between the behavioural and design paradigms and can be used to understand, execute, and evaluate IS research by combining the two [24]. The three cycle framework is based upon seven guidelines of well-designed research, shown in the following list:

1. **Design as an artefact.** The design science must produce a viable artefact in the form of a construct, a model, a method, or an instantiation. In this research, the main goal is the creation of the model-driven smart contract development method, aiming to mitigate the current problems surrounding the field of smart contract development.

2. **Problem relevance.** The objective of design science research is to create solutions to important and relevant business problems. Smart contract development is a novel topic in which a code-centric approach is often chosen over a model-centric approach. The creation of a model-driven smart contract method brings a structure to the development process, potentially preventing loss of funds through faulty or vulnerable contracts. Furthermore, the model-driven approach to development supports the communication between participants of the development process.
3. **Design evaluation.** The evaluation process in design science research is a crucial component in which the utility, quality, and efficacy of a design artefact must be rigorously demonstrated via well-executed evaluation methods. The utility of the model-driven smart contract development method is demonstrated through a use case, after which the quality of the method will be evaluated through an experiment. This way both the utility and the theory behind the method are accounted for.
4. **Research contributions.** The design science research should provide a clear contribution in the areas of the design artefact, the design construction knowledge, or the design evaluation knowledge. This research contributes by providing a method for the model-driven development of smart contracts, as well as providing an evaluation methodology for the comprehension of models. More on the research contribution is reported in chapter 1.2; Contributions.
5. **Research rigor.** Rigor is derived from the effective use of a knowledge base, which consists of theoretical foundations and research methodologies. A literature study on smart contracts, smart contract development, and MDE is combined with situational method engineering to create the method. The usefulness of the method is then assessed through a case study. Furthermore, an experiment is conducted to investigate the claimed benefits of the method through a number of evaluation qualities.
6. **Design as a search process.** In developing an artefact, the research should take the knowledge base into account and make extensive use of existing grounded theories and knowledge to create a solution. The artefact will be created using situational method engineering, in which a method base containing method

fragments is used to create a method tailored to the problem at hand. This will be done on the basis of a literature review.

7. **Communication of research.** Design science must be presented to both a technology-oriented as well as a management-oriented audience. By presenting this research in the form of a thesis, the contents should be understandable for a variety of audiences. The structure of a thesis allows for every step of the research process to be comprehensively explained, and for all the concepts to be defined in a manner the intended audiences can understand.

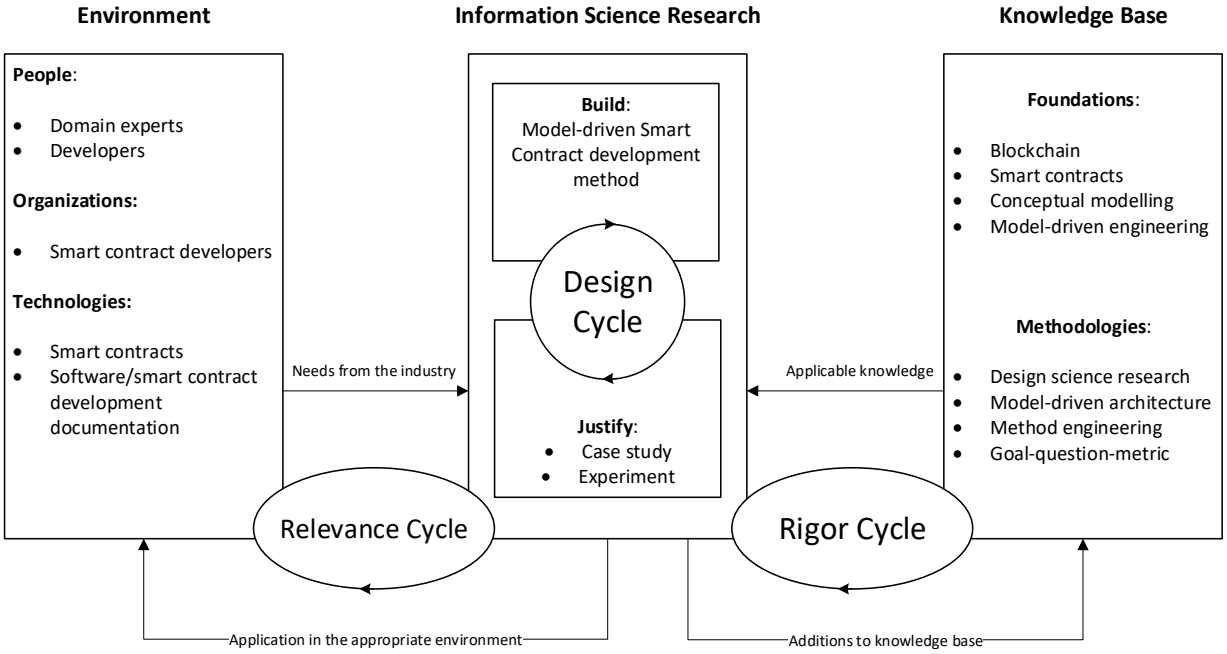


FIGURE 1: TAILORED DESIGN SCIENCE FRAMEWORK

Figure 1 shows how the seven guidelines fit in the design science research framework. This framework is tailored toward the creation of the model-driven smart contract development method. The environment consists of domain experts and developers, which are involved in the creation of a smart contract. All organizations that make use of smart contracts in any form can benefit from this model-driven development method. The technologies used are smart contracts, MDE technologies, and documentation technologies for the creation of smart contracts.

The foundations for this research lie in blockchain research, smart contract research, conceptual modelling research, and MDE research. The methodologies

applied are design science research, described in this chapter, model-driven architecture (MDA), described in chapter 3.4, method engineering for creating the method, described in chapter 4, and the goal-question-metric approach to experimental research, described in chapter 6.

The knowledge base and the environment jointly contribute to the information science research in the middle column, in which the artifact is build. The artifact, the model-driven smart contract development method, is then evaluated using a case study and an experiment. The design science is not solely aimed at the creation of an artefact but has to with the systematic creation of knowledge about, and with, design [26]. The whole process works iteratively, in the sense that every aspect is revisited throughout this research project and the sole focus is not entirely on the creation of the artifact.

2.3 Literature Research Protocol

Blockchain and Ethereum are relatively new phenomenon, so a lot of the research on it is fairly recent. This makes it difficult to determine the relevance of papers based on for instance the amount of references. For this reason, a 100% structured approach to the gathering of relevant literature was not the best fit, and therefore a semi-structured approach was chosen.

The tentative initial set of papers was determined in two different ways. For the literature on MDE the proceedings of the International Conference on Model Driven Engineering Languages and Systems (MODELS) were consulted. For the literature on blockchain and Ethereum, papers were selected based on the citations of the initial description of Ethereum, the 2014 paper by Gavin Wood [27]. The assumption here is that literature touching upon the subjects Ethereum and smart contracts will reference this paper. From the papers found, the relevance was manually determined.

From the start set, keywords were distilled for both blockchain and Ethereum, and MDE. The search for peer-reviewed related work on blockchain and Ethereum, and MDE was done through several online databases, such as Google Scholar, IEEE, ACM, ScienceDirect, and Springerlink. The keywords used in these queries are as follows:

"Blockchain" "Ethereum" "Smart Contract" "Smart Contract Development" "Solidity" "Model-Driven Engineering" "Model-Driven Development" "Model-Driven Architecture" "Transformations" "Model-driven Software Engineering" "Domain-specific Language" "Code Generation"

Keywords that yielded too many results were used in combination with other keywords, for instance:

"Smart Contract" AND "Transformations"

A distinction was made between research on smart contract development and research on the technological specifications of blockchain. Technological specifications are for instance the throughput and latency of a blockchain platform. These technological specifications of blockchain are beyond the scope of this thesis. On the papers found through the queries, both backward- and forward snowballing has been applied [5]. In backward snowballing, the reference list of a paper is reviewed and papers are selected through the review of abstracts. In forward snowballing, papers were selected which cite the initial set papers. Forward snowballing was used because a lot of papers are relatively new and are only findable through this method. The final inclusion of papers was done based on the full paper, and the process was iterated until no new relevant papers were found.

3. Theoretical Background

This chapter describes the knowledge discovered through literature research, described in chapter 2.3. A distinction between three main concepts has been made in order to create a holistic overview of the concepts that are relevant for this thesis. As the smart contract has gained tremendous momentum with the emergence of blockchain, this concept is first explained by describing the main technologies that enable blockchain platforms in their current form (chapter 3.1). With the understanding of what blockchain technology is, the smart contract will be defined and discussed (chapter 3.2). Hereafter, MDE will be discussed (chapter 3.3), as well as a framework for MDE (chapter 3.4). Lastly, a look into existing research combining the concepts of blockchain, smart contracts, and MDE will be discussed (chapter 3.5).

3.1 Blockchain

Blockchain is best known as the technology that runs the Bitcoin cryptocurrency. It is a public ledger system maintaining the integrity of transaction data [28]. The technology was first applied when this cryptocurrency was launched and described by the pseudonymous Satoshi Nakamoto [1]. Swan roughly divides the emergence of blockchain in three phases, namely blockchain 1.0, 2.0, and 3.0 [28]. The first phase is the blockchain supporting a cryptocurrency, the second phase incorporates smart contracts, and the third phase lowers the entry level by combining the blockchain with decentralized apps (dApps). Firstly, the 1.0 as the backbone of blockchain will be explained to create a thorough understanding of the basics. After this, the second phase, the application of smart contracts, will be discussed. The third phase is of little relevance to this research, so will not be discussed further.

So, the first and currently biggest application of blockchain technology is the Bitcoin cryptocurrency. Cryptocurrencies differ from classical forms of currency in many areas. Table 1 denotes a comparison between the two, based on [2].

TABLE 1: COMPARISON BETWEEN CRYPTOCURRENCY AND FIAT CURRENCY

Characteristics		Bitcoin	Fiat currency
Issuance/ management	Issuer	• Automatically issued by the system	• Governments
	Manager	• Managed by P2P network participants	• Governments
Value	Issuance cap	• Specified (21 million BTC)	• None
	Grounds for value	• Trust in the system	• Trust in the government
Money transfer	Transfer time	• 60 minutes on average	• Depends on the sum and distance
	Transfer fee	• Small amount	• Expensive
Privacy	Anonymity of transactions	• Transaction records are clear but anonymous	• High anonymity
	Disclosure of transactions	• Full disclosure	• No disclosure

As can be seen in Table 1, in contrast to the classical fiat currencies, Bitcoin runs automatically without the interference of a central authority. The near collapse of banks deemed too big to fail in 2008 showed the world the fragility of the classic concept of money. Bitcoin is an answer to the non-transparent system of the fiat currency approach and aims to not only minimize the transaction costs, but to let the users themselves safeguard the integrity of the currency [29].

Blockchain is the name of the public decentralized transaction ledger on which the digital currency Bitcoin runs. A ledger, in its essence, is a combination of two things: a list of accounts who own an amount of something, and a list of transactions from one account to another. This way a ledger denotes a proof-of-ownership for a certain good and all the transferals of proof-of-ownership of the goods, which combined can proof ownership at all times. When keeping track of the balance of a cryptocurrency these two components alternate, so there is an initial state, a number of transactions, which in turn result in a new found state. This way the ledger of a cryptocurrency such as Bitcoin can be seen as a state transition system in which the blocks denote the transition and the newfound state [27]. This process can be seen in Figure 2, in which the initial state has 6 accounts owning 10, a transition with three transactions, which results in the new state. If an account does not yet exist

in the initial state (left), it will be created in the newfound state (right), which can be seen in the creation of account G.

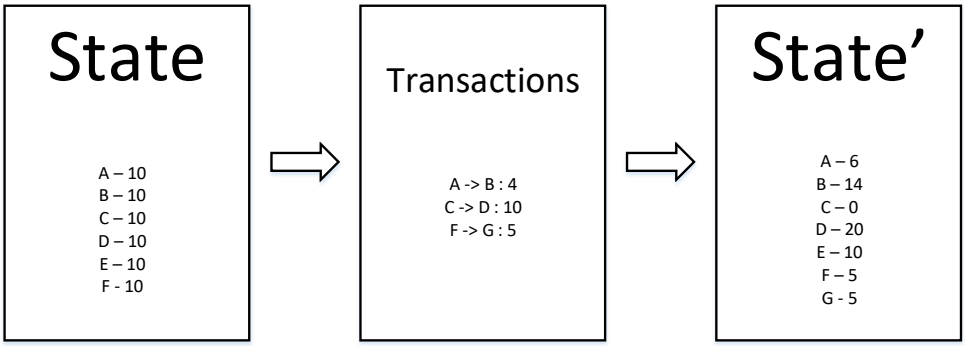


FIGURE 2: A CRYPTOCURRENCY AS A STATE-TRANSITION MACHINE

Classically, a trusted third party keeps track of a ledger and consensus is reached through an aristocratic system, like the fiat currency in Table 1. The biggest problem of cutting out the trusted third party is the question of who will maintain the integrity of the ledger. The problem of maintaining trust among unknown peers is illustrated by the Byzantine Generals Problem, a thought experiment which illustrates the pitfalls and design challenges of attempting to coordinate an action by communicating over an unreliable link [30]. For instance, in the state transition example in Figure 2 the transactions are only valid if the balance in the initial state is greater than or equal to the transaction amount. If a participant in the network were to create invalid transactions and add them, the state would be corrupted. If this is the case, the problem of reaching consensus about the state among participants in a network arises.

In an attempt to solve this problem, blockchain was described. Blockchain combines public-key cryptography, cryptographic hashing, peer-to-peer networking and consensus mechanisms to create a decentralized autonomous ledger [31]. A basic understanding of these technologies and their concepts is needed to understand how a blockchain functions, so the next sections will outline these technologies after which the combined working in a blockchain will be discussed.

3.1.1 Public Key Cryptography

In order to maintain the integrity and confidentiality of accounts while maintaining availability, a system of encryption and decryption must be applied. Data is encrypted to be unreadable for unwanted parties and decrypted to be shown to the desired parties. When using a single key to encrypt and decrypt, the number of keys grows rapidly as the number of users grows. The problem of handing over a massive amount of keys was solved by Diffie and Hellman [32], who first proposed the idea of public key cryptography in 1976. In this cryptographic method a user has two keys: a public key and a private key. The user may distribute its public key, because a key only encrypts or decrypts data. The keys work as inverts, so data encrypted by a public key can be decrypted by a private key and vice-versa. Deducing one key from the other, however, is effectively impossible [33]. Data can be safely shared as long as the sender shares his public key in advance and his private key maintains secure. This process is shown in Figure 3.

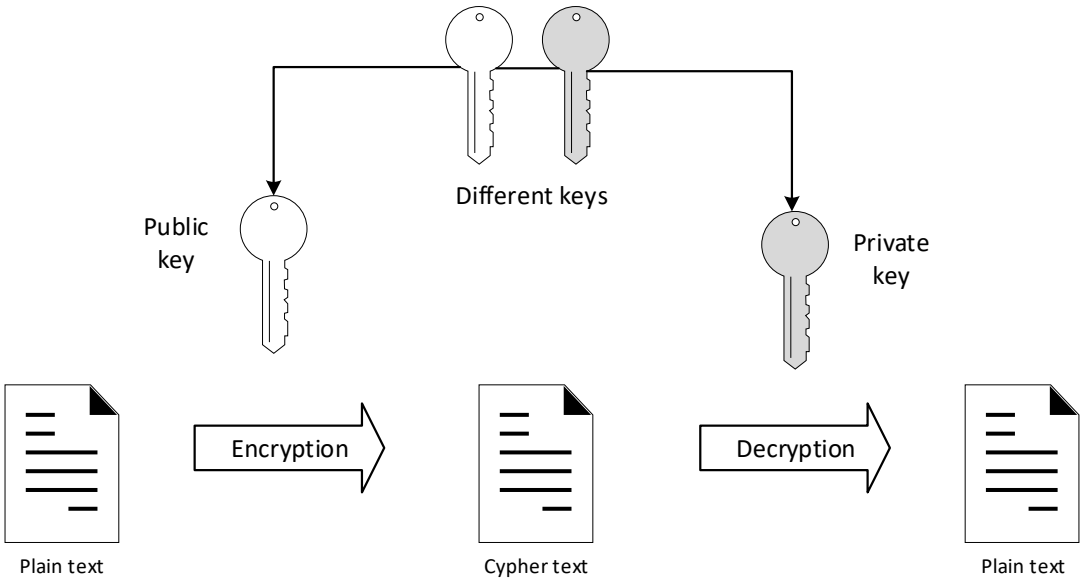


FIGURE 3: ASYMMETRIC ENCRYPTION [34]

Blockchain is not about sending secret messages, so what has asymmetric cryptography to do with any of it? With the private and public key, a digital signature can be established. This is a mechanism which works as proof that a message originates from a sender [35]. In this mechanism, the sender encrypts his data with his

private key and sends it to the recipient. The recipient, in turn, can decrypt the data with the public key, which proves it originated from the sender. This process can be seen in figure 4.

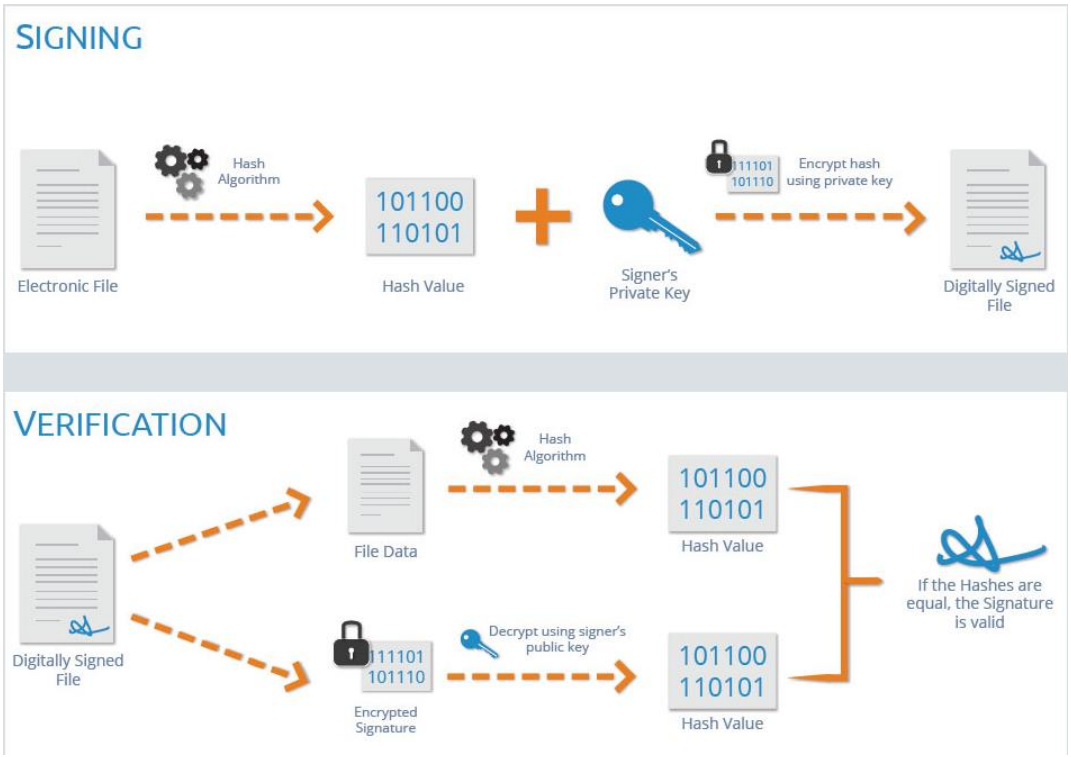


FIGURE 4: DIGITAL SIGNATURES [36]

In a blockchain the public key functions as an address for the person who owns an amount of BTC. This address is stored on the blockchain and is publicly available to everyone with access to the blockchain. The digital signature is used to confirm that the actual address sent a transaction. The hash value of the transaction is encrypted with the private key of the sender. Decrypting this should deliver the same hash value as the hashed transaction, and if not the transaction is invalid. Everyone on the blockchain has access to the public key/address, so everyone is able to verify the transactions.

3.1.2 Cryptographic Hashing

Cryptographic hash functions play a fundamental role in modern cryptography. A hash function maps bitstrings of an arbitrary finite length into strings of a fixed length called the hash-value, or simply hash. Such a function must be a one-way function,

which means the input cannot be deduced from the output. It also must be collision resistant, which means it is computationally infeasible to find two inputs which produce the same output [37]. Input will always result in the same output, but a slight change in the input will produce a completely different hash value. Figure 5 shows this process, along with 2 examples.

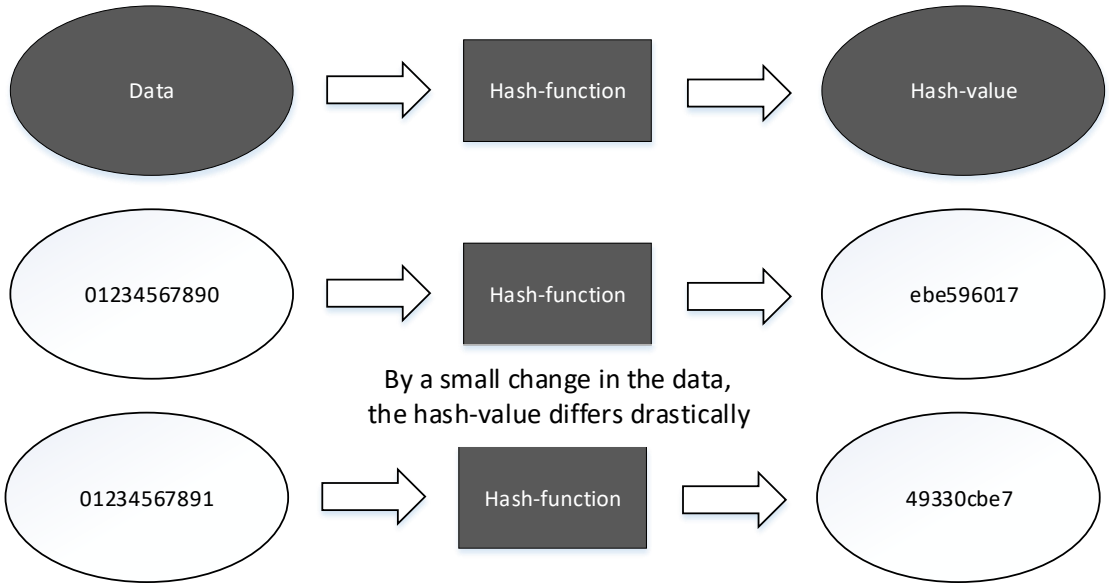


FIGURE 5: CRYPTOGRAPHIC HASHING

Hash-values, in turn, can also be hashed. In the blockchain, all transactions are hashed. It is time consuming and computationally expensive to check the entirety of the hash-list, which is why Merkle trees are used. A Merkle tree is a tree structure and a generalization of the hash list. Each leaf node is a hash of a block of data (in this case a transaction), and each non-leaf node is a hash of its children [38]. The hash of the Merkle tree will alter completely if any data is altered, which is illustrated in Figure 6. This allows for efficient verification. In the blockchain, this is the mechanism behind the detection of falsified data, and it guarantees the continuity and creation of blockchain data through proof-of-work; see chapter 3.1.4.

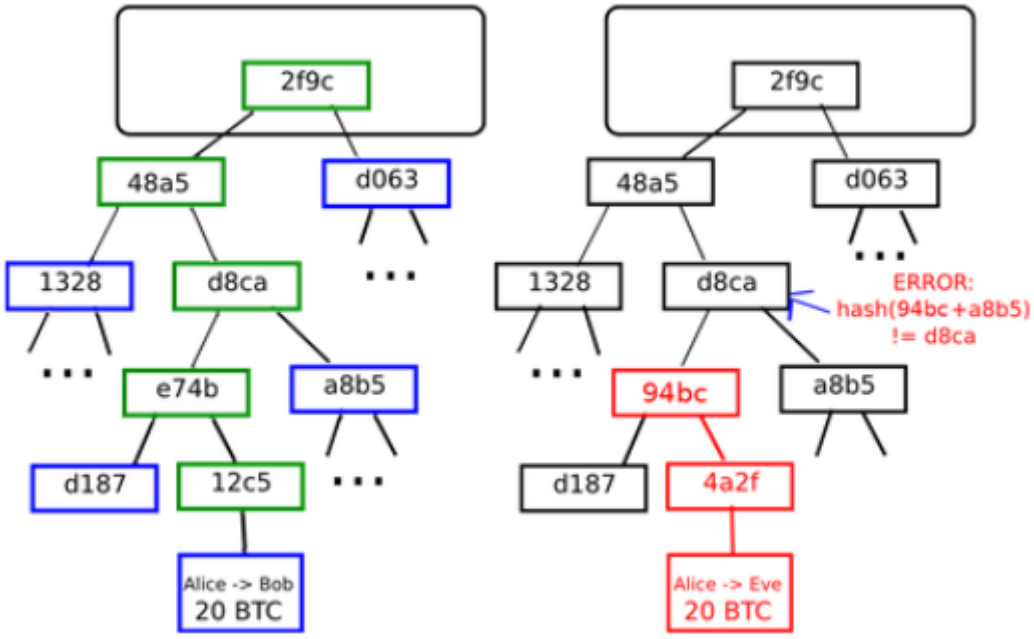


FIGURE 6: HASHING IN A MERKLE TREE

3.1.3 Peer-to-peer Networking

Peer-to-peer denotes a network in which nodes create an autonomous network wherein data is requested and provided among these nodes on equal footing. A node is a physical/virtual machine that communicates via TCP/IP and UDP with other nodes [39]. The role of a node in this network is not fixed as a client or server. This is in contrast with the classic client-server structure in which one party is in charge of the provision and preservation of the data. This one leading party is the centralized server, and the clients request and obtain their data from this centralized server.

By not having a client/server structure, some aspects of the network become more complex. For instance, how data is distributed and the method of data transmission between peers is to be considered [40]. In order for a blockchain to function properly, it is evident that the network is available for all nodes and that propagation functions fluently. Peer-to-peer networking technology is used as a base technology to form the distributed network and eliminates a single point of failure [41]. It also plays a crucial role in the verification and creation of blocks which are added to the blockchain [42].

3.1.4 Consensus Mechanisms

In a classical centralized architecture, a single authoritative database is the source of information, which defines the true data. In a blockchain, all the nodes have a copy of the ledger, and together they decide what the true single state of the ledger is [43]. There are a number of mechanisms which enable the nodes in a network to do so. What most of the consensus mechanisms have in common is that they give the nodes an incentive to keep track of the state in the form of the currency that the blockchain holds.

The Bitcoin blockchain currently runs on the proof-of-work mechanism. Proof of work (PoW) is a mechanism which makes an action more costly. In this scenario the action to be performed is relatively easy to do and making it more costly means making it harder to do. For instance, in 2006 it was proposed as a mechanism to be added to email. Sending spam mail is a relatively easy task, and the proposed mechanism to be added to email would make sending mail more costly, so that spamming would no longer be economically attractive [44]. In the blockchain this process is called mining and is centered about earning a reward. The reward is an incentive to do computational work in order to verify and control the blockchain. To earn the reward, every block is accompanied by a computational puzzle: the data content of the block combined with a nonce must result in hash smaller than a certain value. A nonce is any given value, and the certain value depicts the difficulty of the block. A hash function is pseudo-random so it is impossible to deduce the nonce from the data, so the only way to solve the puzzle is to try all nonce values until the right result is guessed. Solving the puzzle adds a block to the blockchain, which is called mining. In Bitcoin this task is designed in such a way that approximately every ten minutes a new block is added. In figure 7 the proof-of-work mechanism is illustrated. The nonce is incrementally upped until the calculation result is below a certain threshold. In this case the calculation result should be below 100000. This threshold denotes the difficulty of the computational puzzle, the lower the threshold, the harder the puzzle in theory is. The difficulty is adjusted in such a way that the time it takes to mine a block stays the same on average.

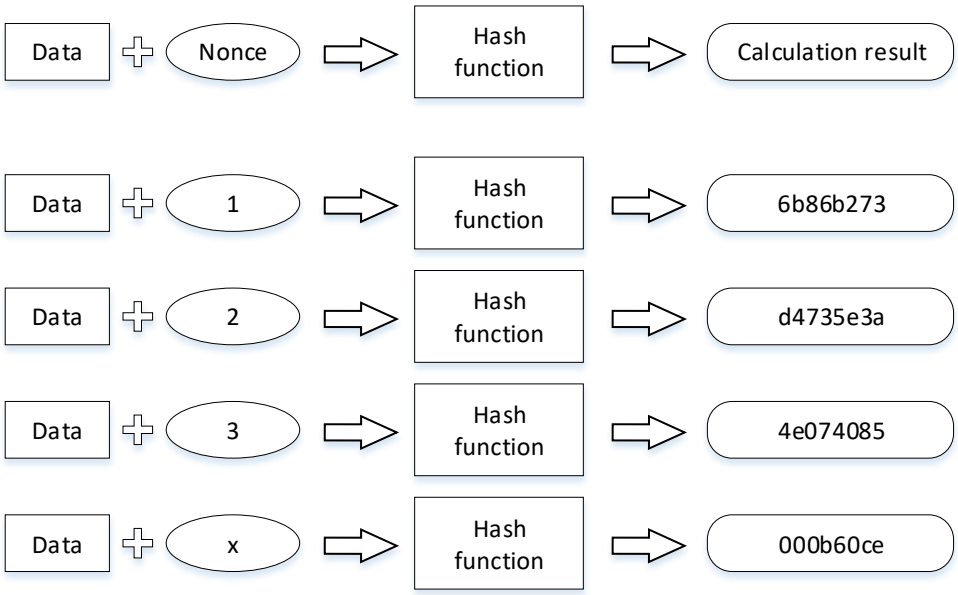


FIGURE 7: THE PROOF-OF-WORK CONSENSUS MECHANISM

The first way to earn a reward is to be the quickest to mine a block. However, to ensure that miners check the quality of blocks, a reward can be taken away if someone proves the mined block to be faulty. This quality-reward is an incentive for people to check mined blocks and not flawlessly adopt the latest blocks to earn the speed-reward. If the nodes mutually accept a block, the collection of transactions in that block is added to the blockchain, and the process restarts with the transactions that were not included in the last block. In Bitcoin, the reward started at 50 BTC and halves every 210,000 blocks (approximately four years).

The top 10 largest blockchain protocols, of which Bitcoin is the largest, are all currently running on the proof-of-work mechanism [42]. Next to usage with malicious intent, which will be discussed later, the mechanism is dependent on energy consumption because of its reliance on computational power. This makes it costly to mine blocks, and as the reward halves every 4 years the incentive to mine is lost over time. A solution to this is to raise transaction fees, but this means blockchain platforms still rely heavily on energy consumption. This is why a number of other consensus mechanisms have been proposed. A number of these mechanisms and the basis for their consensus are shown in Table 2.

TABLE 2: THE CONSENSUS MECHANISMS OF BLOCKCHAINS

Consensus mechanism	**The consensus is based on**
Proof-of-Work (PoW)	Computational work
Proof-of-Stake (PoS)	Ownership of currency
Proof-of-Activity (PoA)	Currency discovery
Federated Byzantines Agreement (FBA)	Majority voting system

The first block created through mining is called the genesis block, after which more blocks are appended. The total series of blocks created through a consensus mechanism is called a blockchain. Table 3 shows a typical block structure with an example and explanation of what the element are. All elements, except for the transactions, are part of the block header. The transactions form the block body.

TABLE 3: STRUCTURE OF A TYPICAL BLOCK

	Element	**Example**	**Definition**
Block header	Block version	02000000	Indicates which set of block validation rules to follow
	Parent Block Hash	Hash	A 256-bit hash value that points to the previous block
	Merkle Tree Root	Hash	The hash value of all the transactions in the block
	Timestamp	24d95a54	Current timestamp as seconds since 1970-01-01T00:00UTC
	nBits	30c31b18	Current hashing target in compact format
	Nonce	Fe9f0864	A 4-byte field, which usually starts with 0 and increases for every hash calculation
Block body	Transactions		

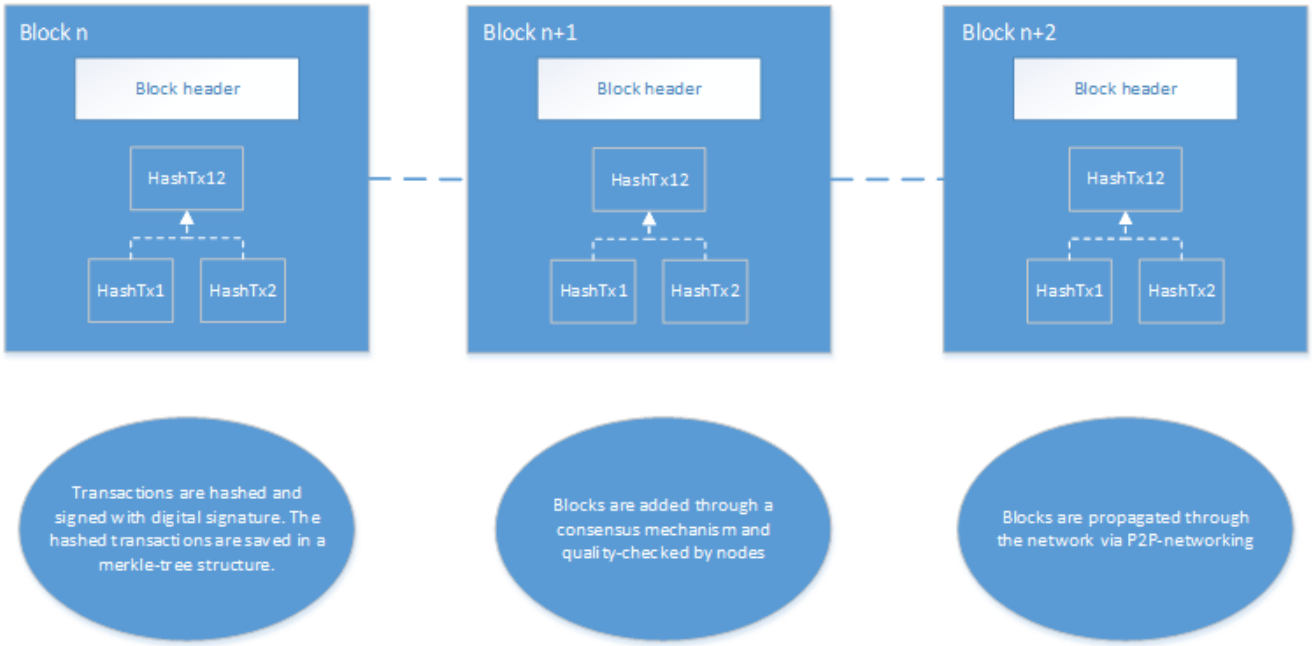


FIGURE 8: BLOCKCHAIN AND ITS COMBINED TECHNOLOGIES

With this block structure in mind, a holistic overview of a blockchain can be made, shown in Figure 8. In this simplified version there are only two transactions per block. A consensus mechanism makes it possible for the parties to reach a single source of truth which is delegated over the blockchain network. However, in these mechanisms there is a reasonable chance that two or more nodes find a valid block almost simultaneously. In this case the other nodes in the network are working on different branches of the blockchain [45]. This happens very frequently, so instead of a linear build-up of blocks, the creation of the blockchain looks more like a tree with a lot of different branches. To decide what the right branch is two principles are being enforced: the principle of the longest branch, and the principle of the heaviest branch. The first one is self-explanatory, the branch with the most blocks will be mined on. In the case of proof-of-work this works out most of the time as in case of a simultaneous mining, the next block decide the main branch [46]. If this does not give a definitive answer to what the main branch is the weight of the branches is decisive factor. Every block has a certain difficulty based on the threshold of the proof-of-work puzzle. The sum of these difficulties provides the weight of a branch and the one with the highest weight is chosen.

3.2 Smart Contracts

As introduced in the previous chapter, the blockchain is a great platform to facilitate trade in the form of cryptocurrencies. Through the immutability provided by the hashing of transactions, the linking of the blocks created by a consensus mechanism, and the propagation of the blocks, a central party is deemed unnecessary. A next step in the application of this technology is the subsidizing of more complex agreements. Code that is stored, verified, and executed on a blockchain is called a smart contract [47]. The idea of a smart contract was proposed by Nick Szabo in 1997 [9]. The main aim of such a contract is to automatically execute the terms of an agreement once certain conditions are met. Simply stated, it is a computer program which follows an *if this happens then that* structure [48].

Smart contracts stand out from traditional contracts in the sense that they carry low legal and transactional costs, and can lower the bar of entry for users. The consumer deals directly with the movement of valuable currency, so the security of such a contract is very important. If you transact currency into a buggy contract, you will most likely lose it [15]. On the blockchain, a smart contract holds digital assets which are released once certain arbitrary conditions are met [49]. For instance, A will transfer an X amount of currency to B, once he receives X currency from C.

In the context of this research a smart contract is a program that runs on the blockchain and has its correct execution enforced by the consensus protocol. Although smart contracts could theoretically serve as entire software applications, most applications lie in the financial or notary category [50]. These match the old definition of contracts, in which a contract is a legally binding or valid agreement between two or more parties. The main objective of such a contract is to fulfill a certain goal and to safeguard against undesirable outcomes, together referred to as contract robustness [9]. Other applications of smart contract are for instance games, but these contracts are far more likely to be developed by people with a far-reaching knowledge of Solidity than financial or notary contracts [50].

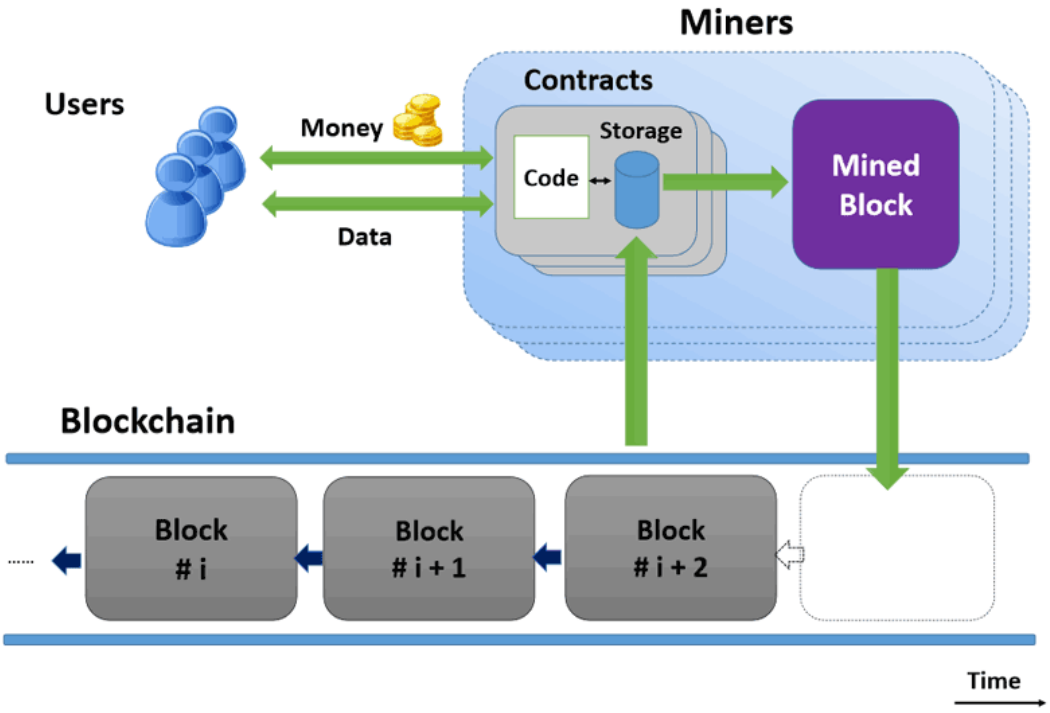


FIGURE 9: SMART CONTRACTS ON THE BLOCKCHAIN [15]

A contract consists of a code, an internal storage, and an account balance. The state of a contract consists of the contract's balance and the internal storage. The state is updated every time the contract is invoked. Figure 9 shows such a contract. The users invoke the contract by sending transactions and data to the contract address, note that the 'money' in Figure 9 can in this case mean any kind of (crypto)currency through a transaction. The miners treat the contract transactions in the same way normal transactions are handled. In the contract code, every time the account receives a message its code activates, allowing it to read and write to internal storage and sent other messages or create contracts in turn. The contracts should not be seen as something that will be fulfilled, but more as an autonomous agent that always executes a specific piece of code when receiving a message, keeping track of its own balance and their key/value store to keep track of persistent variables.

In most distributed computing networks like the blockchain platforms, security measures limit the input and output of external data [48]. For certain smart contracts to function, however, external data is needed. For this reason a distinction between deterministic and non-deterministic smart contracts is made. A deterministic smart contract does not depend on information other than information on the

blockchain. An example is a lottery contract, in which contenders send an amount to the contract. When a certain limit or time is reached, the contract executes a function which randomly picks the winning lottery number and the prize is distributed to a contender. In this contract, no data is used, except for data and information already on the blockchain. Logically, this results in a deterministic contract always having the same output if the input is not changed.

A non-deterministic contract does depend on outside information. This outside information is called an oracle [48]. An oracle provides information from outside the system that the system itself cannot acquire. A classic example of this would be a sports-betting smart contract, in which the oracle is the outcome of the game/event. The smart contract pays out funds based on the outcome of a game/event which calls for a trusted party to provide this outcome.

The development and execution of smart contracts can be done with different blockchain protocols in mind. Bitcoin, for instance, supports a rudimentary scripting system, but this is not very user-friendly. There have been attempts to design applications using the Bitcoin scripting language [51] [52], but this seems too difficult for the average user. It is also very limited regarding the complexity of contracts. With the limitation of the complexity in mind, NXT was created. NXT has templates which can be combined to create smart contracts. Templates still limit the complexity of smart contracts (e.g., not Turing-complete).

Ethereum [49] currently is the most advanced smart contract focused blockchain platform. This blockchain protocol aims to solve the fundamental limitations that Bitcoin has with its scripting language. Ethereum was built primarily with the aim to store and execute smart contracts. This is because Ethereum supports Turing completeness feature that allows creating more advanced and customized contracts. Turing-completeness means it could theoretically be used to solve any computational problem. Ethereum stands out because unlike Bitcoin, it was created with the aim to not only be a cryptocurrency but to be an alternative protocol for building decentralized applications, an overview is shown in Figure 10. For this reason, it has become the blockchain platform on which by far the highest amount of smart contracts are deployed [14].

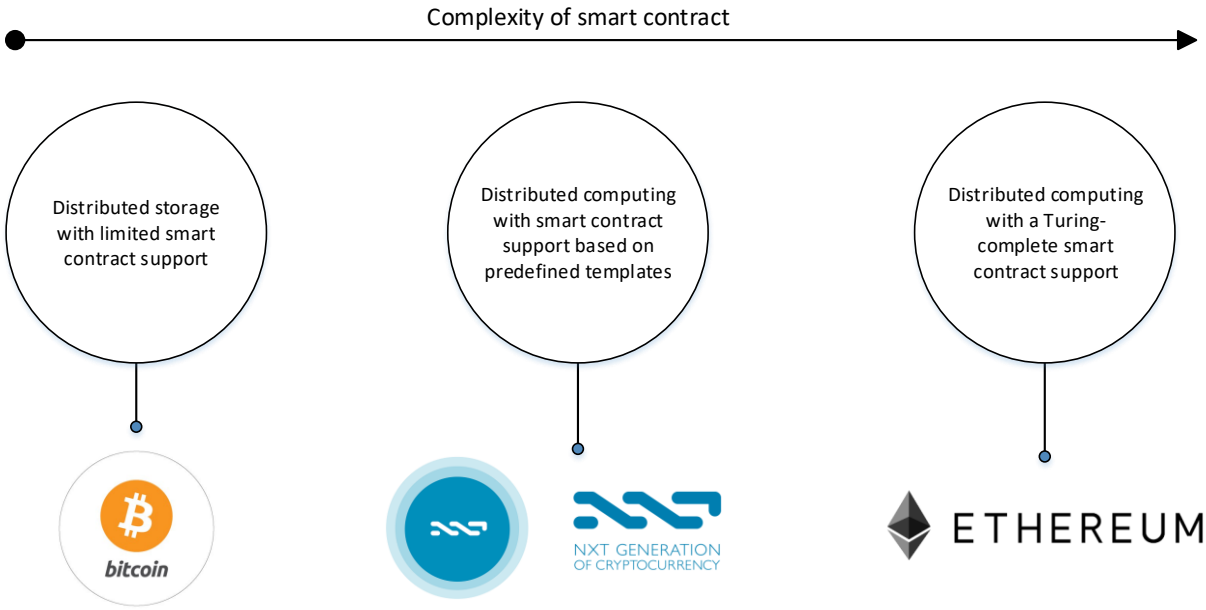


FIGURE 10: SMART CONTRACT SUPPORT IN DIFFERENT BLOCKCHAIN PROTOCOLS

On Ethereum the state is comprised of two types of accounts. The externally owned accounts, controlled by their private keys, and contract accounts, controlled by their contract code [27]. An externally owned account has no code and can be compared to normal accounts on a blockchain with an address and a balance. These accounts can send messages by creating and signing a transaction.

As is the case with Bitcoin, the processing of Ethereum transactions is distributed and is done through the proof-of-work consensus mechanism. An essential difference is that in Bitcoin, all transaction require the same amount of computational power. In Ethereum, the smart contracts call for different amounts of computational power. To ensure that the miners are rewarded fairly for their computational efforts, an additional fee was added to Ethereum transactions. This fee is called gas. Each instruction in the Ethereum bytecode costs a pre-specified amount of gas. When a contract is invoked, the sender of the transaction must specify how much gas he is willing to provide for the execution of the contract (gasLimit), as well as the price for each gas unit (gasPrice). This way the node who does the computation is rewarded the gasPrice multiplied by the pre-specified amount of gas for the execution of the contract. If this exceeds the gasLimit, the execution is terminated with an exception and it will not be added to the blockchain. When such an exception is thrown, the sender still has to pay the gasLimit he specified to prevent resource-

exhaustion attacks. Ethereum can be thought of as a distributed computer platform in which anyone can run code by paying for the associated gas charges.

The amount of gas is determined through a summation of the bytecode expressions that are executed in a smart contract. The bytecode is executed on the nodes through the so-called Ethereum Virtual Machine (EVM). The EVM-bytecode is the lowest level of abstraction in the Ethereum programming paradigm; however because this is incredibly hard to read and write by developers, higher level programming languages have been created for Ethereum. The most widely used is Solidity, a language library similar to C and javaScript. Solidity is the most supported and maintained language, but other languages include Serpent, based on python, and LLL, based on Lisp.

Alharby and van Moorsel did a systematic mapping study into blockchain-based smart contracts. They identified four key issues, namely, codifying, security, privacy, and performance issues [53]. For this research, the codifying and security aspects of smart contracts are especially important. Codifying issues entail difficulties writing correct smart contracts [15], the inability to modify or terminate smart contracts [54], a lack of support to identify under-optimized smart contracts [13], and the complexity of programming languages [55]. These codifying problems in combination with the necessity to do so correctly because of their publicly available nature hinder mainstream adoption and acceptance of the technology. The security issues identified enhance this hindrance, but are more on a technical than a pragmatic level. This can vary from a dependence of the order or timestamp of a block to the way exceptions and re-entrancy is handled [16].

A large portion of the problems with smart contracts and smart contract development could be pinpointed to the lack of formalization of this relatively young field. The lack of standards and best practices makes smart contract development prone to problematic practices. Evidence of this is the study into how much smart contracts are vulnerable to one of the before-mentioned vulnerabilities in which 68% of the contracts had such a weakness [16]. Among these vulnerable smart contracts is the decentralized autonomous organization (DAO), a smart contract which aimed to work as a decentralized hedge fund. An exploited vulnerability in the code allowed a hacker to extract a massive 60 million dollars from the total of 150 million capital

stored in the smart contract. The lack of research in the field of software development practices in the development phase of the smart contract contributes to a large amount of faulty smart contracts.

3.3 Model-Driven Engineering

With smart contract development, it is hard to say what happens to the complexity of software. Currently, the smart contracts are not as complex as regular software systems, but the Turing-completeness of the Ethereum blockchain provide the possibility for increasingly complex smart contract applications. An IT environment is characterized by rapidly changing business requirements, heterogeneous middleware platforms, and the need to incorporate legacy systems with new applications and technologies [20]. This can also be said for blockchain environments, because of their novelty and the rapidly evolving blockchain platforms and their programming languages. Currently, Ethereum combined with Solidity is the most used blockchain platform and programming language for smart contracts, but because of the high volatility, this trend could quickly shift.

In the practice of software engineering, there is a variability in the extent to which the engineers use models in the development phase. On the one extreme, there is the model-centric or model-driven development approach, in which models are used to describe the structure and the behavior of the system, which is then used to generate source code for the system. On the other extreme, there is no usage of models at all, and the focus is solely on the code, hence a code-centric approach. The spectrum of the approaches is shown in Figure 11 [56]. The use of models is applied to lower the level of abstraction and to separate components of a system. It can lower the complexity and create a better understanding of the system.

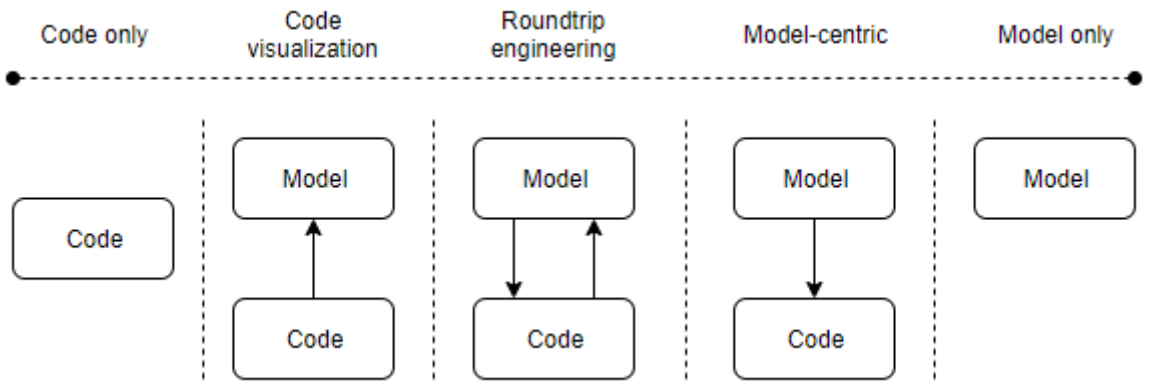


FIGURE 11: SPECTRUM OF SOFTWARE DEVELOPMENT APPROACHES

In a volatile development environment in which for instance the platform or the programming language changes frequently, it has proven to be useful to use a more model-centric approach to increase the re-use of knowledge [20]. One of the most important aspects, the business knowledge, can be re-used and only the implementation of the business knowledge has to be reworked. In a code-centric approach, the whole process has to be reiterated. Smart contract development is a relatively novel field, and its development environment is highly volatile. For this reason, it may be more efficient to look into a model-centric instead of a code only approach. A methodology aimed at a model-centric approach is Model-Driven Engineering (MDE) [19]. This is a software engineering approach consisting of the application of models and model technologies to raise the level of abstraction at which developers create and evolve software, with the goal of both simplifying and formalizing the various activities and tasks that comprise the software life cycle [57].

To elaborate further on what models can mean for the development of software, it is essential to define what a model is. To distinguish models from other artifacts like requirements and data, Stachowiak [58] defines three criteria to be met by a model, elaborated on by Ludewig [59]:

- **Mapping criterion**: There is an original object or phenomenon that is mapped to the model. This original object or phenomenon is referred to as "the original";

- **Reduction criterion**: The model does not map all the properties of the original, but only the relevant properties. It is, however, needed that the model mirror at least some of the properties of the original.
- **Pragmatic criterion**: The model can replace the original for some purpose. This is referred to as being useful.

These three criteria make the concept of a model very broad. Beizvin even states that everything can be seen as a model [60]. Kuhne defines models in the context of MDE as "an artifact formulated in a modeling language, such as UML, describing a system through the help of various diagram types" [61]. So in the context of this research, a model is a pragmatic artifact that maps an original object or phenomenon while creating an abstraction to focus on the properties that are most important.

MDE can be applied to a development process for a variety of reasons. It can improve the quality assurance of system requirements [62] and add a level of formalization and standardization to the system development process [63]. With the models and transformation rules, automation can be applied to generate code [64] or do model-based simulation to assess the quality [65]. MDE can be implemented to improve the communication and information sharing between stakeholders and within the development team [66]. Using models in the development phase can also ease the porting of solutions to new platforms [67].

In the literature, there exists some ambiguity surrounding concepts in the model-driven field. In this research, a distinction between three levels is made. First, MDE serves as the umbrella term for the research area in which the gap between the problem domain and the software implementation domain is reduced through the systematic transformation of problem-level abstractions to software implementations [68]. Abstractions, or models in this context, serve three main purposes: they generalize specific features of real objects, classify the objects into coherent clusters, and can aggregate objects into more complex ones [69]. Second, Model-Driven Development (MDD) is a development approach that uses the abstractions or models as the primary artifact of the development process. In MDD, automation is used to generate code from formalized structures or models [19]. Through this automation in the development process, the software quality could improve, the complexity can be reduced, and the higher level of abstraction could make development more accessible

to a broader audience [57]. In some research, MDE is described as MDD [68], but here MDD is the application of MDE. MDE describes the entire Software Development Lifecycle (SDLC), and MDD focuses on the model to implementation transformation. Third, the Model-Driven Architecture (MDA) is an approach to MDD proposed by the Object Management Group (OMG). It is a set of standards for the execution of an MDD process [70]. The relation between these three concepts can be summarized as follows: MDA is an approach to MDD, which is a subset of MDE (Figure 12).

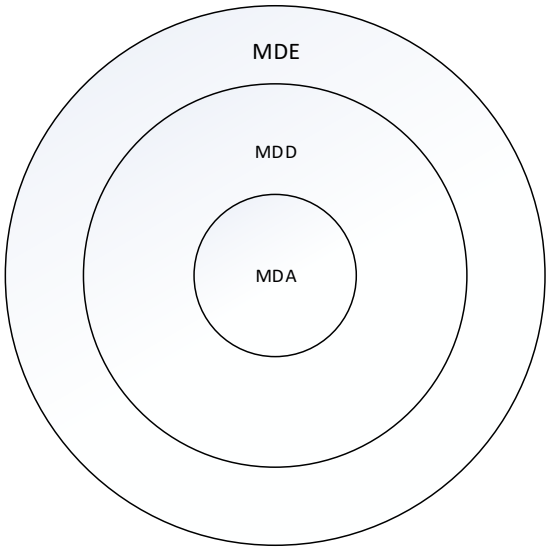


FIGURE 12: RELATION BETWEEN MDE, MDD, AND MDA

There are other frameworks for MDD, like the Eclipse Modeling Framework (EMF) [71] or the Java Metadata Interface (JMI) [72]. MDA is chosen here because in the literature it is named as a perfect case for explaining MDE concepts, as all the standard phases of a software development process such as analysis, design, and implementation are appropriately supported; second, given the importance of the OMG in the software industry, MDA is currently a reference conceptual framework adopted in many organizations [69]. Furthermore, the abstract view of the system is represented through the OMG's modeling standards, chapter 3.4, which allow transformations to major open or proprietary execution platforms [56].

3.4 Model-Driven Architecture

Model-driven architecture (MDA) is an instantiation of the MDD approach. MDA is a framework based on UML and other industry standards for visualizing, storing, and exchanging software designs and models [73]. MDA is model-driven because it provides a means for using models to direct the course of understanding, design, construction, deployment, operation, maintenance and modification. All artifacts such as the requirements specification, architecture descriptions, design descriptions, and code are regarded as models. It is a framework by the OMG [70] that suggests the use of models and transformations of these models in the software development process. Its name is somewhat misleading as it is not an architecture, but a standardized approach to MDE based on abstraction of platform similarities [74]. The primary goal of MDA is interoperability between tools and the long-term standardization of models in popular application domains [75]. MDA distinguishes between the Computational Independent Model, the Platform Independent Model, the Platform Specific model, and code:

- **Computational Independent Model (**CIM): This model describes the business system and is sometimes referred to as the domain model. It contains no details of the system and is specified by domain experts, using vocabulary that is familiar to the practitioners of the domain in question. It often contains the application's business functionality and behavior through use case and activity diagrams, and the actors that interact with the application. This model can be seen as a contractual element that acts as a reference to check if the requirements are correctly fulfilled by the other models;
- **Platform Independent Model (PIM)**: This model is a view based on the CIM in which the system is described from a platform-independent point-of-view. By doing so, the PIM models the system in a way suitable for different platforms of similar type. The goal of the PIM is to realize logical data, establish dependencies and define workflows. It requires that model elements contain enough information so that logic implementation and code generation can be made possible;

- **Platform Specific Model (PSM)**: In this model, the system-view and its specification details are tailored for the use on a specific platform. In this model, the PIM is transformed into a code model, which in turn can be transformed into source code. The difference between source code and a code model is that source code is a succession of textual lines, and code models are a structured representation of the source code;
- **Code**: In this 'model' the PSM is transformed to executable source code tailored to the PSM's specified platform.

Figure 13 shows the relation between these four concepts and an example application. The first phase of an MDA process is gaining the business knowledge and requirements. However, in many practical applications of the MDA, this phase and the consequent modeling into the CIM are skipped. The PIM is then taken as an initial model. This phase is argued to be highly essential and assists in thoroughly understanding the business knowledge [76]. The introduction of techniques and models in this phase help close the gap to the stakeholders' world and way of thinking.

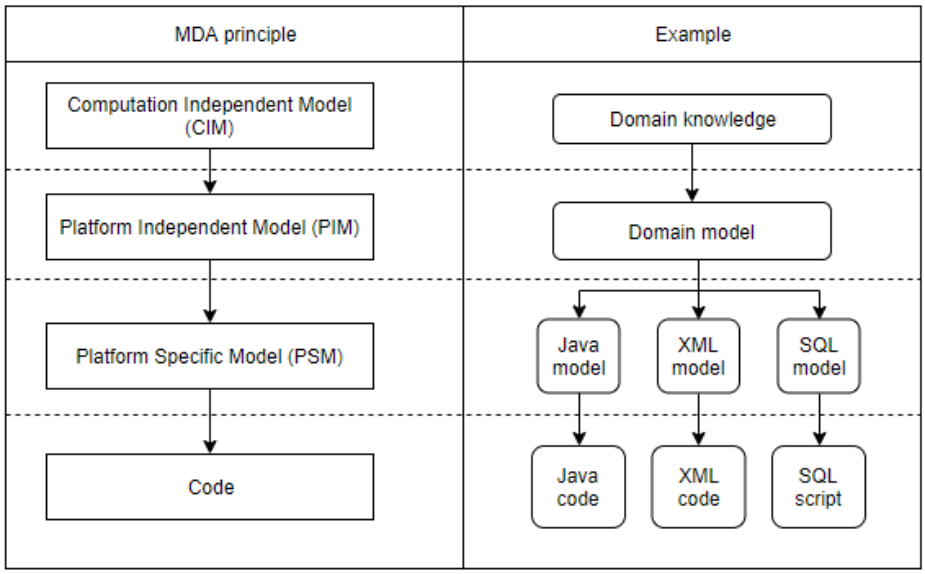


FIGURE 13: MDA PRINCIPLES

One of the critical features of this framework is the transformation between the models. Transformation rules describe how a model in a source language (the source

model) can be transformed into one or more models in a target language (target model), as shown in Figure 13. Model transformation relies on a set of mapping rules between models. These mapping rules are based on knowledge about the application domain or the implementation technology. MDA provides the foundations to develop tools that implement these transformations, which in turn can ensure the consistency and validation of models.

The MDA framework is based on industry standards that are supported by the OMG. These standards aim to formalize the specification of models through meta-models and to better the communication about models. The base MDA standards are the Unified Modeling Language (UML), the Meta Object Facility (MOF) and the XML Metadata Interchange.

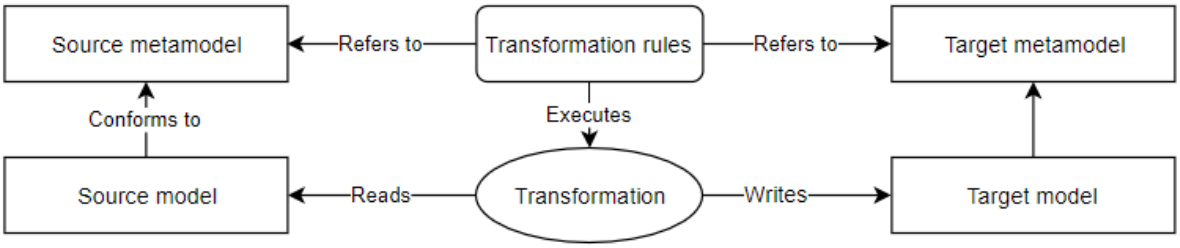


FIGURE 14: TRANSFORMATIONS IN MDA

The Unified Modeling Language is a language for visualizing, specifying, and documenting software systems [77]. The UML is a result of the best practices in modeling engineering and can be used to describe complex systems through models [78]. The graphical representations of models in UML are called UML diagrams, which can represent different views of a system.

Models can also be used to describe other models, which is called meta-modeling. The OMG describes a meta-model as "a model that defines a modeling language and is also expressed using a modeling language" [79]. In theory, there are endless meta-models, because every model could be described by another model. The OMG describes the MOF (Figure 15), in which the meta-modeling is limited to the

meta-metamodel. The MOF aims to stimulate the development and interoperability between models and metadata driven systems [79].

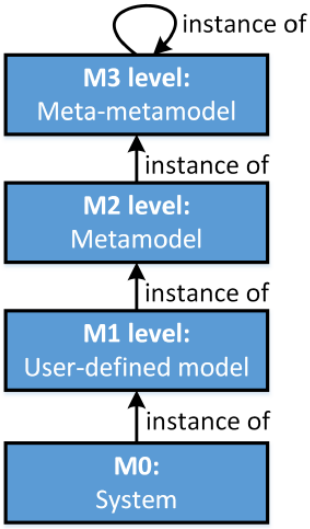


FIGURE 15: MOF STANDARD

The MOF is a standard for meta-modeling definitions. The lowest level of this architecture is the system, which is a real-world instantiation of a user-defined model (M1 level). This user-defined model is an instance of a meta-model (M2 level), which in turn is an instance of a meta-metamodel (M3 level). UML can be seen as an example of a meta-model that is captured in the MOF formalism.

3.5 Model-Driven Engineering Approaches to Smart Contract Development

Smart contracts supported by distributed ledger technology is a 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 contract development. In this chapter three of these attempts are described extensively. The included MDE approaches have one or more modeling language(s) to support the concerns and viewpoints associated with a smart contract, at least the PIM to PSM transformations, and mappings are supported through a rationale.

3.5.1 Agent-Based Approach

The first approach is described in the paper "From Institutions to Code: Towards Automated Generation of Smart Contracts" by Frantz and Nowostawski [80]. It is based on the concept Grammar of Institutions by Crawford and Ostrom [81], which lies in the area of institutional analysis. This Grammar of Institutions is used to decompose institutions into rule-based statements. These statements hereafter are compiled in a structured formalization.

The grammar of institutions finds its roots in agent-based modeling (ABM). ABM is a computational modeling paradigm in which phenomena are modeled as dynamical systems of interacting agents [82]. The model consists of a set of agents that encapsulate the behaviors of the various individuals that make up the system, and the execution consists of emulating these behaviors [83]. Often the behavior of the agents is modeled through sets of statements in which the behavior becomes explicit. In this case, the statements are constructed from five components, jointly abbreviated to ADICO:

- **Attribute**: The actor's characteristics or attributes;
- **Deontic**: the nature of the statement as an obligation, permission or prohibition;
- **Aim**: the action or outcome that the statement regulates;
- **Conditions**: The contextual conditions under which the statement holds;
- **Or else**: Describing the consequences associated with non-conformance to the statement.

Using these components, statements on the execution of the smart contract are made. In the description of this method, an example of a voting system is given: "People (Attribute) must (Deontic) vote (I) only once (C)." The statements are then linked by the structure of nADICO [84], 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. By modeling the relationship of the agents in a rule-based manner, ABM make it possible to identify interdependencies between different human activities in a system.

The set of prescriptions is then transformed into a contract skeleton which has to be finished manually. Frantz and Nowatiszki [80] provide a DSL which facilitates

the mapping from the statements to the skeleton contract. It is argued that the provision of a contract skeleton separates the specification task from the implementation, ensuring no crucial functionality is forgotten. Furthermore, it is argued that the Grammar of Institutions invites non-technical people to the smart contract development process.

3.5.2 Process-Based Approach

The second approach is described in the paper "Untrusted Business Process Monitoring and Execution Using Blockchain" by Weber et. Al. [85]. This approach is aimed at business processes across organizations. In the paper, the focus is on supply chains, in which smart contracts are used to address the lack-of-trust problem in collaborative business processes. A business process specification like the Business Process Modeling Notation (BPMN) [86] is used as the PIM and it is transformed into Solidity code in order to provide an automated and immutable transaction history, to provide a mediator in the process control logic, and to obtain an audit trail for the complete collaborative business processes.

The translator, which does the transformation, takes a business process specification as input and generates smart contracts. It is called at design time, so it may not be known which actor is assigned which role. This is why the output of the generator is a factory contract which holds all information needed for instantiating the process. The overall translation is done in two phases. First, the business process is iterated through, and for all elements a list of the previous and the next elements are determined. Each element is then translated to Solidity with its respective links. The transformation is based on workflow patterns, so the business process has to follow one of five basic control flow patterns or else it cannot be translated. This means not all elements of for instance BPMN are suited for transformation.

In a later paper [87], Weber et. Al. build upon their work on business processes to smart contracts. Here, they present another translation method which involves the transformation from a BPMN model to a petri net, which then is translated to Solidity code. This manner of transformation negates the need for the business process specification to follow a certain pattern but does involve more steps.

3.5.3 State Machine Approach

The third approach is described in the paper "Designing Secure Ethereum Smart Contracts: A Finite State Machine Approach" by Mavridou and Laszka [88]. This approach is based 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 [89]. The aim of this approach is threefold. First, it aims to provide a formal model with clear semantics in which smart contracts can be modeled, which decreases the semantic gap and eliminates issues arising from it. Second, the clear semantics allows the connection from its framework to formal analysis tools. Third, the code generator allows developers to implement smart contracts with a minimal amount of error-prone manual coding.

The PIM in this approach is the Finite State Machine (FSM). This mathematical model of computation can be in exactly one of a finite number of states at any given time. The FSM changes state in response to external inputs or to a specified timed transition. It consists of a list of its states, its initial state, and a collection of conditions for transitions. The transformation of the FSM to Solidity is partly automated, because to ensure Solidity code quality, some manual coding might be necessary. Properties that cannot be modeled in FSM can be added through plugins, which are aimed at implementing patterns and countering vulnerabilities.

An FSM should contain an initial state, a finite set of states, and a set of transitions between these states. The transitions are activated once certain conditions, called guards, are met.

In Table 4, an overview of the three approaches, namely the grammar of institution approach (ADICO), the collaborative business process approach (BPS), and the Finite State Machine approach (FSM), is given. In Table 4, the approaches are compared based on the following metrics, loosely based on the selection criteria described by Krogstie [90]:

- **Based on concept:** The approaches have different application domains for the smart contract. ADICO uses the concept of institutions to describe a smart contract, where actors have a deontic quality. BPS is based on collaborative

business processes with multiple actors who have their subset of tasks. FSM is based on the smart contract as a state machine. This does not limit the approach to one certain application domain.

- **Approach scope:** The distinction between academic and industry is made here. ADICO and FSM have an academic scope because they do not explicitly describe an industrial application. BPS is aimed at collaborative business processes, which are deeply rooted in industry.
- **Target users:** This metric is about who is going to use the approach. ADICO claims that their structured natural language allows non-technical users to participate in the development process, but the final development phases need to be supported by technical people. BPS is aimed at industry, and the target users are participants in collaborative business processes. Their users will be organizations which are collaborating in for instance supply chains. FSM is an open approach, so the users can vary strongly. The automated generation of the code can appeal to non-technical, as well as well-grounded developers.
- **PIM and PIM structure:** This states what the Platform Independent Model of the approach is. ADICO's model is a collection of structured natural language statements which contain the elements described above. BPS is modeled in a business process specification like BPMN. BPMN is a standard in business process modeling, so this is a standardized approach. In the follow-up paper, the BPS is transformed into a petri net, which can be seen as a PIM but also as a part of the transformation. In this case, we chose the former. FSM uses the FSM model, which is a variation on the state machine diagram described in UML. This is also a standardized modeling language.
- **Model transformation and direct transformation:** This describes if the transformation from PIM to Solidity is automated, semi-automated, or manual, and if it with or without an intermediate step. Automated does not need interference from the developer, semi-automated calls for adjustments of the output, and manual means no automation in the transformation process. The ADICO is semi-automated because the model transforms the nADICO statements into a skeleton contract. The BPS is automated and semi-automated. In the first instance, the business process specification is transformed fully

automated, given that the specification follows one of five pattern flows. In the follow-up article, the business process specification is first transformed into a petri net, after which the transformation is automated, so this is a semi-automated process. Only this BPS is indirect, because of the transformation to petri net. The other transformations do not have an intermediate step so are direct. The FSM transformation process is fully automated. There may be some manual coding afterwards, but this is to improve the existing code, meaning that this transformation process is marked as automated.

- **Result of the transformation:** Here, the output of the transformation process is described. All the approaches output Solidity code, but the completeness of this output varies. ADICO outputs a skeleton contract, meaning that it provides a parts of a contract, which the user then has to manually complete. BPS outputs a factory contract, meaning that it describes the collaborative workflow and the different participants can add their own code to this factory contract. Both the follow-up to BPS and FSM output a complete smart contract. Again, some manual adjustments might be necessary, but the output could theoretically be launched onto the blockchain.

Lastly, the **advantages** and **disadvantages** are denoted. These are formed from the viewpoint of creating a smart contract MDA process. To summary:

- The ADICO approach is the only approach using structured natural language. The formal syntax makes the statements unambiguous, but the formality also counters the claim that non-technical users will be able to easily employ the ADICO approach. The ADICO model shows more characteristics of a high-level programming language than an easily readable structured natural language. Furthermore, the output of the transformation needs a lot of manual coding compared to the other approaches. The deontic element in the ADICO model does explicitly show what a user must, may, or may not do, making the behavioral aspect of the smart contract explicit. However, it is not explicitly stated how this deontic aspect would be monitored or implemented.
- The BPS approach, both of them, are tailored for collaborative business processes, and in this domain, BPMN is a standard model. This means that in the domain, this approach can be highly applicable. The actors or participants

in the process are explicitly stated, and these users get their rights, but the smart contract functions more as a mediator between these participants than as a centralized contract. This means it is fit for collaborative business processes but less suited in other application domains in which the smart contract is used for more than mediation. In the follow-up of the BPS, the petri net transformation broadens the possibilities of transforming business process specification into Solidity, but this process adds an extra layer of complexity. The linear flow of these business processes make this approach fit for supply chains, but less for recursive contracts.

- The FSM approach treats smart contracts as finite state machines and has a logical modeling language for this approach with the finite state machine. It has a formal syntax, and the transformation into Solidity requires minimal manual coding before implementation. The downside is that finite state machines are relatively complex models, and objects and roles are left implicit in this approach. However, the use of patterns can be used to counteract these disadvantages, and these patterns can also be used to counter known vulnerabilities.

	Frantz & Nowostawski	Weber et al	Weber et al	Mavridou & Laszka
Based on Concept	Grammar of Institutions	Business processes	Business processes	Smart contract as state machine
Approach scope	Academic	Industry	Industry	Academic
Target users	Non-technical and technical	Collaborative businesses	Collaborative businesses	Non technical
PIM	ADICO statements	Business process specification	Business process specification	Finite State Machine model
PIM structure	Structured natural language	Standard model	Standard model	Standard model
Model transformation	Semi-automated	Automated	Semi-automated	Automated
Direct transformation	No	Yes	No	Yes
Result of transformation	Skeleton contract	Factory contract	Smart contract	Smart contract
Advantage	• Natural language can be interpreted by non-technical users; • Formal syntax reduces ambiguity; • The deontic aspect makes the behavior explicit;	• Business process specifications often already exist; • Actors are distinguished; • BPMN is a widely used standard in industry; • Automated transformation;	• Business process specifications often already exist; • Actors are distinguished; • BPMN is a widely used standard in industry; • Automated transformation;	• Minimal manual coding; • Formal syntax; • Non-linear flow through a model; • Smart contract as a state machine make concepts from FSM and smart contract easily relatable; • Usage of patterns against vulnerability;
Disadvantage	• Complex way to structure natural language, almost a programming language; • Not fully automated, development phase is still manually executed; • It is not stated how the deontic element will be evaluated.	• Only a subset of business process specification models can be transformed; • Linear flow, which means that multiple flows are hard to map in one model; • Factory contract is not executable if participants do not comply	• Linear flow, which means that multiple flows are hard to map in one model; • Transformation from BPS to petri net to Solidity is complex; • Aimed specifically at collaborative business processes, negating other usage	• Roles are not explicitly stated; • Objects are left implicit; • Modeling complete and correct FSM's is a complex task

TABLE 4: COMPARISON OVERVIEW OF MODEL-DRIVEN APPROACHES TO SMART CONTRACT DEVELOPMENT

4. Constructing the Method

In this chapter the method is constructed. This will be done through an approach that is called method engineering, which is the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems [91]. A subset of method engineering is situational method engineering (SME), in which the development method is tailored to the project at hand [92]. In this case the project at hand is used relatively loosely, as a method for the development of all sorts of smart contracts is made. SME aims at defining information systems development methods by reusing and assembling existing method fragments [93]. The term method fragment originates from [92], in which the analogy with software components is made. Software components are constructed separately to consequently be combined to form a cohesive software system. Method fragments are formed and defined independently and stored in a method base. New methods can be constructed using these method fragments, using the most appropriate fragments for the situation at hand. SME aims to tailor methods to a specific project at hand in order to increase the productivity of the development, and to better the quality of the product of the method [94]. The SME approach is summarized in Figure 16.

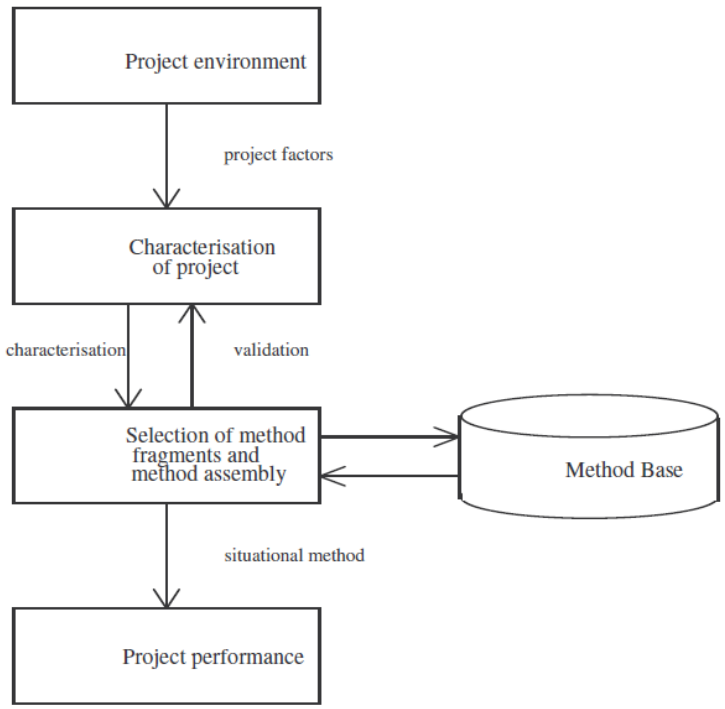


FIGURE 16: THE PROCESS OF SITUATIONAL METHOD ENGINEERING

Based on the SME approach, the MDA framework tailored to smart contract development will be constructed. Distinguishing between the three main components of an MDA framework is important in the construction of the method. The CIM, PIM, and PSM are tailored to different goals in the method engineering process. For this reason, the components are described separately. Following the component description and methods for modeling the components, the transformation between the components is described. A crucial assumption in the method is that the domain knowledge is obtained through a separate process. The domain knowledge creation is beyond the scope of the method.

To create a holistic view of the model-driven smart contract development method, the method fragments will be presented as a process-deliverable diagram (PDD). This is a meta-modeling technique adapted from the UML activity diagram [95] and the UML class diagram [96]. Explicitly describing the activities and the deliverables of the method add a level of formality, as well as a grounded definition of what the method is [97]. The meta-process side of the diagram, the left side, shows the activities and the transitions. The activities can be decomposed into sub-activities, thereby creating a hierarchical activity decomposition. The deliverable side of the diagram, the right side, consists of a concept diagram, which is a variant of the UML class diagram. The connection between the process and the deliverable side is made by a dotted arrow, in which it connects the activity with the adjacent deliverable. For a full specification of the PDD modeling technique, refer to [97]. An important notion to keep in mind is that the PDD is constructed as a linear model, but the smart contract development process can be approached iteratively. This means that every subsection of the process can be revisited and reiterated at different phases of the process. In Figure 17, the PDD of the entire method is modeled at a very high level. Each of the components will be discussed separately in the coming sections, creating an elaborate overview of how the method is to be executed.

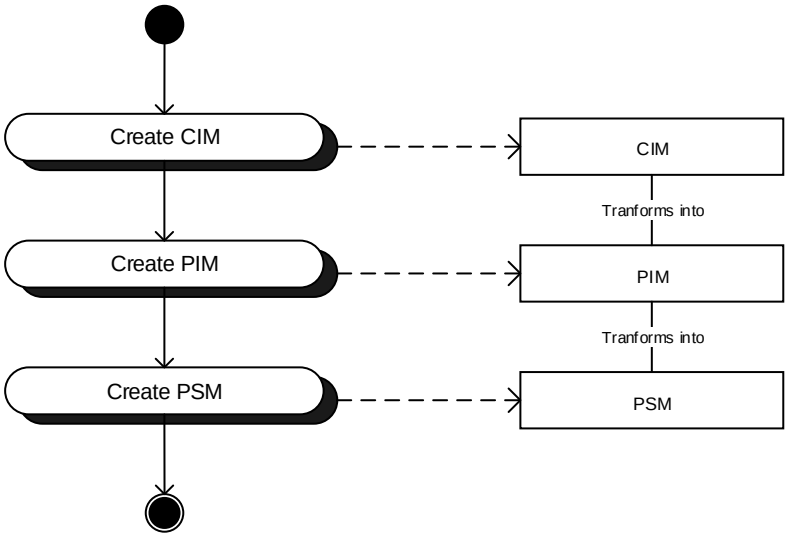


FIGURE 17: HIGH-LEVEL PDD MODEL OF THE MDA FRAMEWORK

First, the requirements and chosen modeling languages for the components of the method are denoted in chapter 4.1. These requirements are based on the literature review and the properties of the components of the MDA framework. Hereafter, for each of the components the chosen modeling technique, the process of creating and evaluating the model, and the transformation rules are described. To support the understandability of the description of the models, a small example of a smart contract is given. This simple smart contract shows a betting smart contract. The description of this betting contract can be found below:

Example betting system: At the start, a user can place a bet on one of two options before the outcome of this event is decided. After the outcome of the bet becomes known, the system either pays out in case of a win or keeps the bet.

In chapter 5, the model-driven smart contract development method is shown in its entirety through a more elaborate case study.

4.1 Method Requirements

To elicit the requirements for the smart contract development method, it is important to explicitly state the goals of the method. The overall goals are to (i) bridge the semantic gap by lowering the threshold for domain experts and (ii) to support developers in the creation of the contract. These goals are a bit high-level but can assist in creating the requirements on a more elaborate level. The choice for a certain modeling language is an important one, as the choice of technique affects the set of phenomena that can be modeled, and may even restrict what the modeler is capable of observing [98].

A distinction between two main roles is made to create the requirements, namely the role of domain expert and the role of developer. The domain expert knows what concepts should be included in the smart contract and the developer has the know-how to translate these domain concepts into technology concepts. The different phases of the MDA framework can guide the communication and interoperation between these two roles in the development process. For each component, it is described how the different roles make use of the models, and the requirements are tailored to these roles.

Smart contracts are a novel way to coordinate interaction between independent entities. A central challenge to a broad application of this opportunity is the unambiguous and correct specification of the smart contract. The goal of the CIM is to guide the transition between the domain knowledge and the specification of a smart contract, without directly focusing on concerns surrounding the implementation.

For the domain expert, this is the modeling step in which in the domain concepts that are essential are elicited from the domain knowledge and to formulate these essential domain concepts in a structured manner. The CIM is a tool for the domain expert to communicate the domain knowledge to the developer. Therefore, the CIM should provide a structured manner in which the domain concepts can be denoted in a clear overview. For the developer, the CIM serves as a blueprint of what the smart contract should represent regarding behavior. The focus is on the domain properties and not on the technical details, so the modeling language should be

suitable to represent this. The requirements for this model can be summarized as follows:

The CIM should provide a high-level overview of the smart contract, which maps the essential domain concepts in a systematic, structured manner and shows the developer all possible interactions with the smart contract.

Use case modeling is a modeling technique that is often used as a way to map the domain knowledge without the inclusion of technical details. The choice not to use this technique in this MDA process is based on the multitude of possible scenarios and the increasing (unnecessary) complexity of the use case model as the possible scenarios increase. The domain experts often do not have modeling experience, so instead of, for instance, use case modeling, an approach which uses text in a structured manner is more appropriate in this setting.

As can be read in 3.5.1, the grammar of institutions has been applied for the model-driven development of smart contracts [81]. Kravic et al. have already laid the groundwork for the application of the grammar of institutions for the development of smart contracts, but in their research, it spans the entire development process. Looking at the requirement goals of creating a structured textual overview of the domain knowledge and having an overview of the possible interactions the actors have with the smart contract, the grammar of institutions fulfills the requirements for the modelling technique of the CIM.

As defined in chapter 3.3, the PIM describes the functionality and behavior of the system in a structured way but does not include the technical specification of the implementation platform. It builds on the CIM, by providing an architecture suited for various platforms. The PIM aims to abstract away from the technical details to validate the correctness of the model, to ease the production for various platforms while maintaining the functional and behavioral specification, and to map the interoperability and integration between platforms more clearly by using platform independent terms.

For the developer, the PIM again serves as a blueprint, but now the functional aspects of the smart contract are included. The developer can create the PIM with the aim of visualizing what the execution of a smart contract will look like. The domain

expert can use the PIM as a way to understand what this execution will be like. This cumulates to a modeling technique which is detailed enough to enable a developer to model the full functionality, while simultaneously being high-level enough to give the domain expert an idea of what the smart contract functionalities are.

In the smart contract programming paradigm, a smart contract is often explained succinctly by stating that it is a series of if-then statements. The smart contract follows a certain path of conditions which result in a certain outcome, defined by an agreement. This way, the programming of smart contract parallels a lot of characteristics of event-driven programming, in which the flow of the program is determined by events. Examples of events are for instance user input, variable triggers like elapsed time, or other programs/contracts triggering it. This leads to the following summarized requirements:

The PIM should provide a functional and behavioral overview of the smart contract in which the notion that a smart contract is comprised of states is evident. The model should be extensive enough for the developer to create a functional overview, and accessible enough for the domain expert to communicate about this functional overview.

With these requirements, it makes sense to model the smart contract as a Finite State Machine. The FSM is a model related to event-driven development in which the FSM is constructed to represent the behavior of a reactive system. The smart contract as an FSM gives a clear overview of the conditions, the transitions, and the possible flows through the smart contract. For this reason, the PIM will be modeled as an FSM model. As is the case with the CIM, the FSM has already been used in research into the application of MDE in smart contract development. Mavridou et al. [88] have described the transformation from FSM to Solidity code, which will be the fundaments for the description of this method fragment.

The platform-specific model (PSM) is not picked by formulating requirements but based on the popularity of the programming language. Solidity is by far the most used programming language on Ethereum and will, therefore, be picked as the PSM. If other programming languages gain popularity, the transformation rules from the PIM to the PSM need to be evaluated and adjusted. The domain knowledge transformed into the PIM, however, will not lose its value by the advent of a new platform or programming language.

4.2 The Computational Independent Model

To tailor to the goals of the CIM, the grammar of institutions is used [81]. As the name states, this was originally defined to better understand what institutions are. The grammar of institutions is a structured template for statements in which the behavior of agents becomes explicit through the interaction participants have with it. By describing an institution in this manner, one avoids the mistake of treating institutions as things that exist separate from the behavior of the participants. Similarly, information systems can be treated as such [80]. In this case, the smart contract poses as the information system, and without a CIM, it can be designed and developed without explicitly stating the possible interactions that participants have with it, which often results in unforeseen usage of the smart contract. Using the grammar of institutions as the CIM can assist in decomposing a smart contract into rule-based statements that explicitly state the interactions and behavior it will encompass.

In the usage of the grammar of institutions, the notion that everything is a model becomes apparent. It is a collection of statements about in this case the smart contract. These statements contain five components, namely Attributes, Deontic, Aim, Conditions, and Or else. Together they are referred to as the ADICO format, taking a letter from each of the components. An ADICO statement can be formally defined as a tuple (A,D,I,C,O), where:

- **A is the attribute:** Attribute is a holder for all participant-level variables. It can range from all participants in a group to a specific subset. If no attribute is mentioned, the statement applies to all members of a group or all participants in the smart contract.
- **D is the Deontic**: The deontic component draws on the modal operations used in deontic logic to distinguish prescriptive from non-prescriptive statements [81]. The set of deontic operators consists of permitted, obliged, and forbidden. If one of the operators is taken as a primitive, the other two can be defined regarding that primitive. This means that if A is forbidden, one is not permitted to do A. In the ADICO format, may (permitted), must (obliged), and must not (forbidden) are used. The deontic component is used to ensure that statements do not contradict each other.

- **I is the Aim:** The aim is the specific action or outcome to which a system refers. One condition to this component is that the aim must be physically possible, as an agent cannot be logically expected to undertake a physically impossible action or effect a physically impossible outcome.
- **C is the Condition**: The condition indicates the set of variables that define when, where, or how a statement applies. If the condition is missing from the statement, it automatically means that the statement holds at all times.
- **O is the Or else:** The final component describes what happens in a situation of non-compliance to the statement. In the grammar of institutions this component is called a threat and for a threat to be qualified as an or else, it must meet three qualifications. These qualifications are largely based on legislation and ask for the or else statement to be (1) backed up by another rule, (2) enforceable, and (3) to be crafted in an environment for the discussing, prescribing, and enforcing of rules. In a smart contract environment, the enforcing of rules happens through distributed consensus.

Statements are made by combining these five components. A distinction is made between shared strategies, which contain attributes, aim, and condition; norms, which contain attributes, deontic, aim, and condition; And rules which contain all five of the components. The or else component needs a complementary statement in which the consequence is explicitly stated. The components of the ADICO framework can be summarized by the deliverable side of the PDD shown on the right in Figure 18, the process of creating the set of ADICO statements is modeled on the left. An example of a set of statements is shown in Table 5.

TABLE 5: EXAMPLE ADICO STATEMENTS

Example set of ADICO statements
A user (A) must (D) place a bet (I) before the event (C) or else the bet will not be valid (O)
A user (A) may (D) not place a bet (I)
A user (A) must not (D) place a bet (I) after the deadline (C)
The system (A) must (D) register the deadline (I) when the event has happened (C)
The system (A) must (D) register the bet (I) when placed (C)
The system (A) must (D) pay out (I) in case the bet was right (C)
The system (A) must not (D) pay out (I) in case the bet was wrong (C)

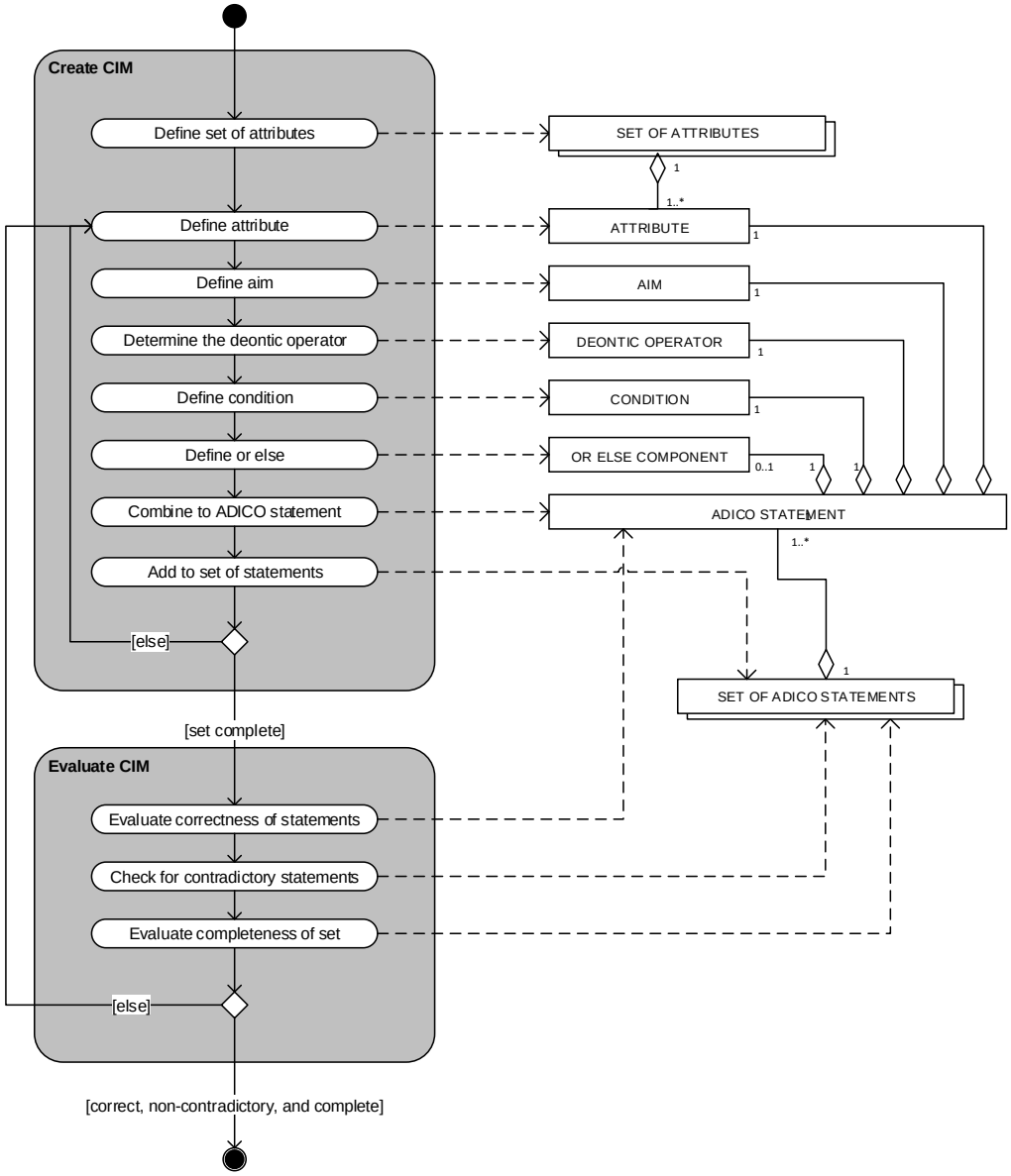


FIGURE 18: PDD OF THE CIM CREATION AND EVALUATION

As can be seen on the left side of the PDD, the first process is to create the CIM. The model in this case is the entire set of ADICO statements. To create this, it is wise to first define a set of attributes, so to consider which actors and subgroups of these actors are stakeholders in the system. Doing so gives a holistic overview of who is going to be using the system, and explicitly defining their attributes can ensure that no viewpoint is overlooked. The deliverable of this definition process is the set of attributes.

Following the creation of the set of attributes, a recursive sub-process is started. First, the aim of the attribute is defined. This aim is a specific action or outcome

that can be possible within the system. After the aim is formed, the choice of deontic has to be made. Is the attribute permitted, obliged, or forbidden to do the aim? This is denoted by may, must, or may not respectively. Then, the condition for the aim to be executed must be defined. This condition can be any set of variables, ranging from when a statement holds (when) to how a statement is to be followed (through a defined process). Lastly, it must be determined if there is an or else component for the statement. In some cases, there is no consequence to non-compliance to the statement. These statements are called norms. For other statements, called rules, non-compliance does have a consequence, and an or else component is necessary. The result of this sub-process is that the five components, or four if there is no or else statement, are combined to an ADICO statement, which subsequently is added to the set of ADICO statements.

The condition for the process of the CIM creation to be finalized is that the set is complete. This is a relatively subjective condition, because a set could, in theory, be always more extensive. In this case, the condition complete means that the statements cover all behavior that is described in the gathered domain knowledge. This should lead to a correct overview of the behavior of the system, and if not, the process needs to be re-iterated. Thus, the end of the CIM creation process is marked by the complete set of ADICO statements deliverable.

The next step in the CIM modeling phase is the evaluation of the model. This process happens in three steps, namely the evaluation of correctness, the check for contradictory statements, and lastly, the completeness is evaluated once more. The correctness of a statement is based on the correct notation of the four or five components. The description of these components shows how these are to be formed. A sentence which mixes the components, or uses different terminology is incorrect and should be adjusted to meet the correctness norms.

If the correctness of all the statements has been established, the set can be checked for contradictory statements. There can be two different contradictions distinguished in this check: the contradiction inside an attributes statements and the contradiction between two different attributes' statements. When a contradiction is inside one attributes' statements, it can be discovered very intuitively using simple logic. An example of this is when an attribute is permitted to do an aim in the one statement and forbidden in the next. Between two different attributes it may become

more complex and context dependent. An example is when two different attributes are obliged to fulfill one aim, and the order of this aim is important. Which attribute fulfills the aim first may influence what happens to the other attributes' rule. The main point of this check is to check that statements do not create a paradox in the system's execution.

The last check that is done is the completeness check again. In the process of checking the correctness and checking for contradictions, statements might be adjusted or removed. It is possible that in this process the completeness of the set of statements is affected. The same conditions as for the completeness check at the end of the CIM creation process is applicable. If one of the three quality metrics is not met, the CIM creation process has to be revisited. If not, the creation of the PIM can start. The transformation between the CIM, which explicitly states the behavior of the attributes in the context of the smart contract, and the PIM, which joins this behavior with the functional aspect of the smart contract, will be elaborated on in the coming section.

4.3 The Platform Independent Model

The next step in the model-driven smart contract development process is the creation of the platform independent model (PIM). The assumptions made by choosing the FSM modeling technique is that the smart contract has states, and in these states, functions allow users or other triggers to change the state. This makes it possible to model a smart contract as an FSM [31]. An FSM in its essence is made up of a set of states and a definition of a set of transitions between these states [88]. A formal definition of the FSM is a tuple (S, S0, T, C), where:

- **S is a finite set of states**: A state is a particular moment the FSM can be in. Functions are provided to invoke actions and change between these states.
- **S0 is the initial state**: The starting state of the FSM;
- **T is the set of transitions**: A transition forces the FSM to take a set of actions if the associated conditions, called guards of the transition, are satisfied.
- **C is the set of conditions**: The conditions or guards which are associated to the transitions, which when they are met can invoke the change between states.

The states of the FSM are modeled by a circle, the transitions by an arrow, and the conditions are modeled under the transition name between brackets, shown in Figure 19.

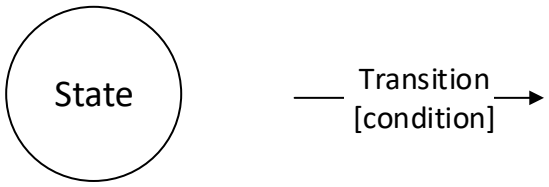


FIGURE 19: THE ELEMENTS OF AN FSM

The transitions between the states are a way to show the behavior of the smart contract. As discussed in the CIM specification, this behavior is something already modeled in the CIM. This is where the transformation between the CIM and PIM comes into play. The transitions of the PIM are based on the behavioral definition given in the set of ADICO statements. This way, the CIM ensures that all the behavior elicited from the domain knowledge is modeled in the PIM, bridging the gap between

the domain knowledge and the PIM. Furthermore, the CIM makes the PIM more understandable, as its transitions are described in structured natural language form, often better comprehensible by non-technical people. The transformation is based on manually transforming the ADICO statements to FSM transitions. Lastly, the set of ADICO statements can provide a basis for a completeness check of the PIM. If behavior modeled in the CIM is missing from the PIM, the model is almost certainly incomplete. The process of creating the FSM is modelled in Figure 20.

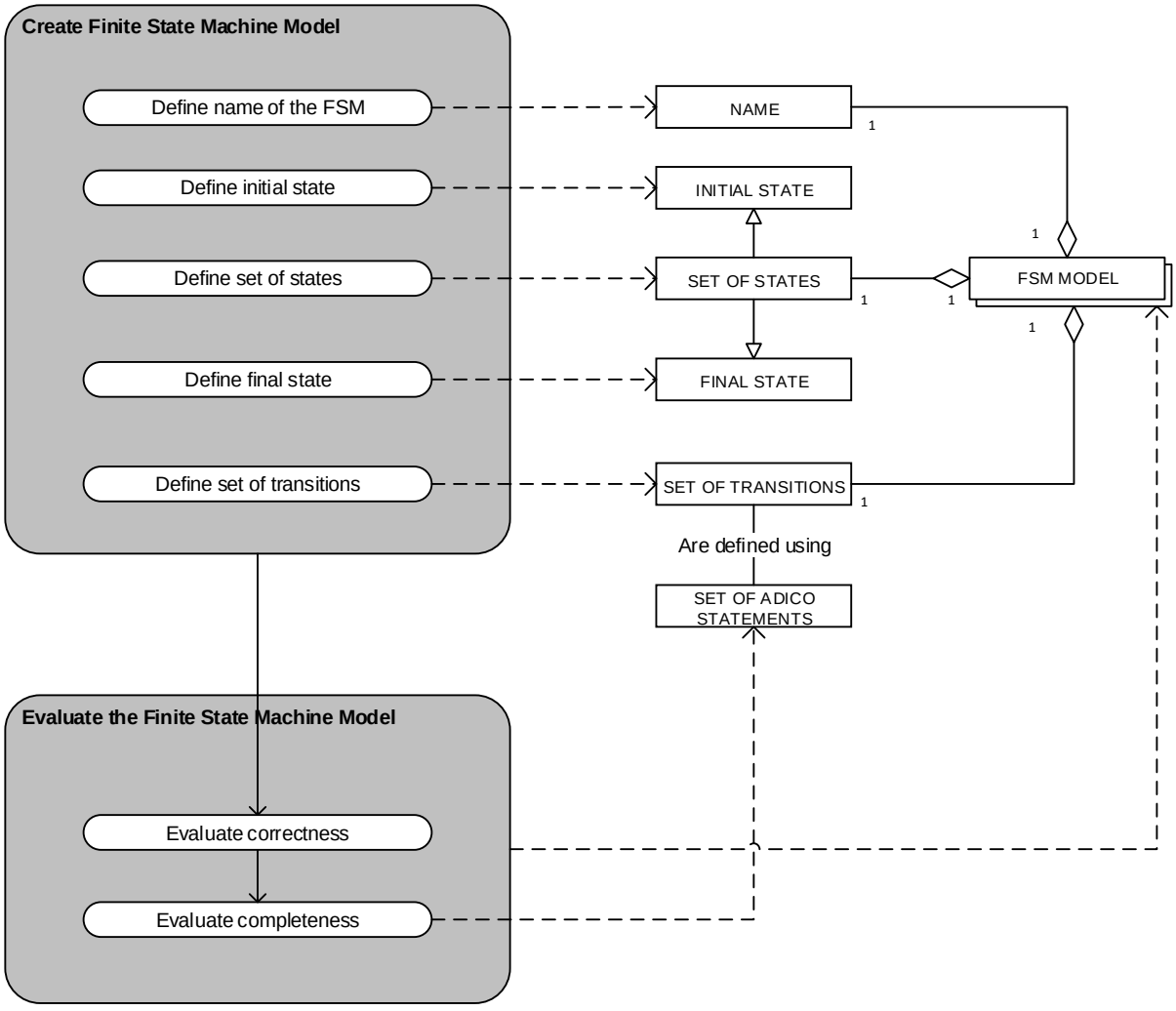


FIGURE 20: PDD OF CREATING AND EVALUATING THE FSM MODEL

The order of the processes modeled on the left-hand-side is not a large concern in the FSM creation process. The set of ADICO statements is used to create a set of transitions. After the model creation, the FSM is evaluated in two steps. Firstly, the correctness is evaluated, meaning that it is checked that the model does not contain mistakes like unwanted transitions between states or unwanted loops. After this the completeness of the model is evaluated, using the set of ADICO statements as an indicator for the behavior it should contain. An example FSM is shown below in Figure 21. The transitions contain numbers between parentheses, based on the ADICO statements, which are shown in Table 6. An overview of the components containing the set of states, the set of transitions, and the set of conditions are shown in Table 7. The ADICO statements from the example in the previous section are used to define the transitions between the states.

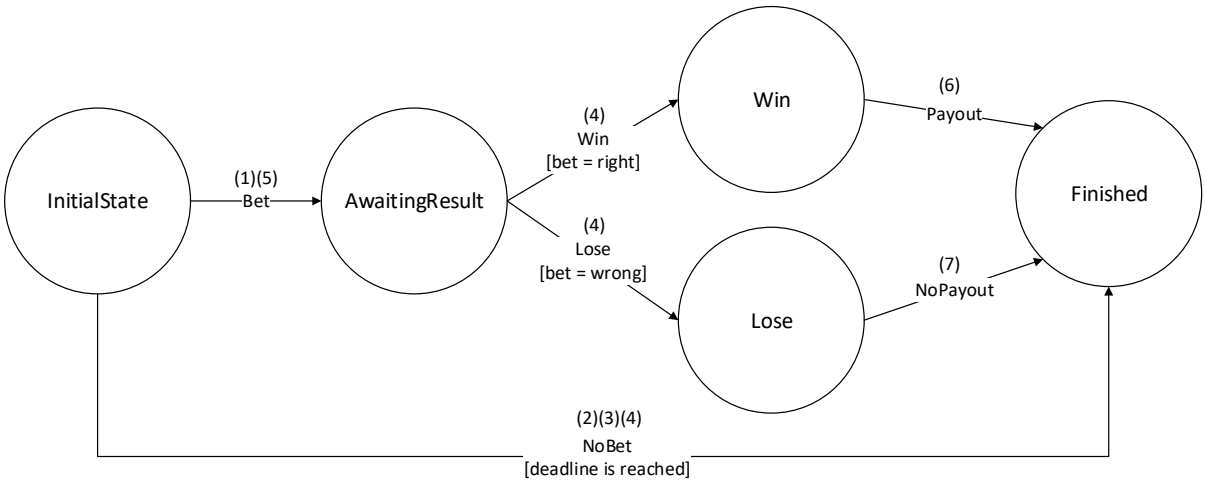


FIGURE 21: BETTING SYSTEM EXAMPLE OF AN FSM MODEL

Example set of ADICO statements
(1) A user (A) must (D) place a bet (I) before the event (C) or else the bet will not be valid (O) (2) A user (A) may (D) not place a bet (I) (3) A user (A) must not (D) place a bet (I) after the deadline (C) (4) The system (A) must (D) register the deadline (I) when the event has happened (C) (5) The system (A) must (D) register the bet (I) when placed (C) (6) The system (A) must (D) pay out (I) in case the bet was right (C) (7) The system (A) must not (D) pay out (I) in case the bet was wrong (C)

TABLE 6: BETTING EXAMPLE SET OF ADICO STATEMENTS

Set of states	Set of transitions	Set of conditions
InitialState	Bet	
AwaitingResult	Win	Bet is right
Win	Lose	Bet is wrong
Lose	NoBet	Deadline is reached
Finished	Payout	
	NoPayout	

TABLE 7: OVERVIEW OF THE STATES, TRANSITIONS, AND CONDITIONS

In order to ready the FSM for a transition to a platform, the definition needs to be extended. This allows for a model to be automatically transformed into a working smart contract. As the PSM in this MDA framework is Solidity, the extended properties are largely based on Solidity properties. If in the nearby future another programming language gains momentum, it would be fairly easy to tailor the transformation process to this PSM. For now, the transformation to Solidity is described. First, the Solidity programming language is described in the platform-specific model section, after which the extension of the PIM and the transformation from PIM to PSM is described.

4.4 The Platform Specific Model

The PSM in this MDA framework is code written in the programming language Solidity, the most widely used and supported programming language for smart contracts. Solidity is a high-level Turing-complete programming language with a syntax similar to that of JavaScript. It is statically typed, meaning that the type of the variable is set, and this type is not changed. It supports inheritance, meaning that an object or class can be based on a parent-object or parent-class, and will display the same implementation as the parent-object or parent-class. In Solidity, contracts are structured using functions and classes in a way that resembles object-oriented programming. It would be too extensive to give a full description of the programming language in this section, but the necessary basis is described.

To give a basic insight into how the code allows a smart contract to be a state machine, three component are important to understand. Firstly, the enum, which is a user-defined type in Solidity. An enum is used to create a list with the set of states. This enum will be used to keep track of which state the smart contract is in. Secondly, the functions, which are used to transition between the states. And thirdly, guards or

modifiers, which are used to define the conditions for a transition between states. Modifiers are defined after which they can be reused throughout the contract, allowing for the definition of certain patterns. The following simple smart contract shows how these components are used.

```
pragma solidity ^0.4.11;

contract StateMachine {
    enum States {
        A,
        B,
    }

    States public state = States.A;

    uint public creationTime = now;

    function nextState() internal {
        state = States(uint(state) + 1);
    }

    modifier timedTransitions() {
        if (stage == States.A && now >= creationTime + 10 days)
            nextStage();
        _;
    }

    function a()
        timedTransitions
        {
        }
}
```

The pragma statement shows which version of Solidity is used, which is important as it undergoes continuous version updates. The contract is called StateMachine, after which the enum 'States' is declared. This user-defined variable has two states, A and B. In the next line the starting state is declared. The variable creationTime is declared next, getting the value now. This means that it represents the time of the launch of the contract. This is in UNIX time, showing the time elapsed from the 1st of January 1970 in seconds. The function nextState makes a transition from A to B possible. In this case, this is done by shifting one in the enum States. If necessary, this can be done specific, by explicitly stating to which state it should transition. However, there are only two states, so this is not necessary here. The modifier timedTransitions describes the condition for a transition between A and B, namely the contract being in state A, and 10 days to be passed. This is then used in function a to make the transition from A to B.

The basics of a state-driven Solidity smart contract are an enum containing the states, functions to transition between these states, and modifiers to create conditions for the transitions. In its essence, that is what an FSM contains. The transformation between the PIM and the PSM can be automated in order to create a Solidity smart contract fundament based on the FSM. The Solidity smart contract begins as a general template, which has four main components. These four main components are the states definition, the variables definition, the patterns used in the contract, and the transitions defined as functions. This way, the smart contract looks as follows:

```
Contract contractName {
      StatesDefinition
      VariablesDefinition
      Patterns
      Transition(t1)
      ...
      Transition(t|→|)
}
```

4.4.1 States Definition

The states definition is the set of states combined in an enum. After the definition of the enum, the initial state is declared. This looks as follows in the smart contract:

```
enum States {S0, … , Sn}
States private state = S0;
```

The states are already defined in the FSM model, so all the states can automatically be transformed into the definition of the enum States. The FSM model also shows the initial state, so this can also be automatically transformed to form the full states definition.

4.4.2 Variable Definition

Solidity is a statically typed programming language, so all the variables have to be declared. This happens in the variable definition. All the variables get a type, an access modifier, and a variable name. It is important that all the variables are declared

because they cannot be used if they are not. The variables need to be entered manually but can be based on the domain concepts mentioned in the CIM. This is also a check for the completeness of the variable definition: if domain concepts are missing from the variable definition, it is incomplete.

4.4.3 Patterns

The patterns which can be implemented in the smart contract are often modifiers which are used throughout the smart contract. For this reason, the patterns are declared below the variables definition so that implementation is made easy in other sections of the contract. Some patterns are applicable per function (timed transition), others are used throughout the entire contract (transition counter). The patterns are a way to counter the security vulnerabilities described in chapter 3.2. The following patterns can be used in the smart contract:

Acces control: In certain smart contracts, it is important that not every node in the network is able to control all the functionalities of a contract. For this reason, transitions (or functions) can be restricted to be only accessible to a certain address. The node who launches the contract is marked as the owner and has restrictive rights.

Locking: The reentrancy vulnerability was the cause of the infamous DAO attack. A function is called within another function and puts the smart contract in an undesired loop. To counter this vulnerability, a principle called locking can be applied. It is a modifier which first checks if the contract is locked. If not, it is locked, the transition is executed, and after execution, it is unlocked again. This way, functions in the contract cannot be nested within each other in any way.

Transition counter: The transaction-ordering dependence is a vulnerability which describes that the state and the values of variables stored in an Ethereum contract may be unpredictable. Due to the decentralized nature, when a user calls a function, he cannot be sure that the state of the contract does not change before this call is actually executed. This is a challenge for smart contracts, as multiple users may invoke a contract at the same time and the order of the execution of these functions is unknown. A solution to this is to implement a transition counter, which enforces a strict ordering on function executions. For every invocation of the contract, the user is asked for a transition number as input. This transition number is incremented with one after each

function execution. This way the user can be sure that the function executes before any other state change happens. This mitigates ordering dependence vulnerabilities and minimizes exceptions due to unpredictable states.

Timed transitions: Treating the smart contract as a state machine, the timed transition is a transition that often recurs. This transition does not have input or output but do have a number specifying the time. If there are a lot of timed transitions in a smart contract, it is wise to implement this through a modifier. This modifier checks before every transition if a timed transition must be executed before the transition. If so, the timed transition will be executed if the time limit has been reached. Writing these timed transitions manually may lead to vulnerabilities, so using this modifier is advised to counter unwanted behavior in the contract.

4.4.4 Transitions

Lastly, for every transition, a function is made. In this function, the information from the PIM is used. A transition has the following layout:

```
function transitionName (type(input1) input1, ...,
type(input(n) input(n))
      pattern(transition)
      {
            require(state == States.transition^from);
            Guards
            Statements
            state = States.transition^to;
      }
```

There is a check if the contract is in the right state, and hereafter the guards and statements are fired. The guards are the conditions described in the ADICO statements and need to be fulfilled in order for the function to be executed. The statements can be a large variety of expressions. Solidity is a Turing-complete language, which means that in theory, every computation is possible. For this reason, the statements and possibly additional guards have to be added manually. After this, the state changes to

the next state of the contract. Based on the FSM model, the creation of the functions can be partly automated. The FSM shows the transitions, what state the transition originates from, and what state it goes to. The process of creating the smart contract foundation is shown in Figure 22. The contract needs to be finished manually but seeing as the smart contract is state-based, the developer only has to program what happens in the transitions. Next, to only having to program the transitions between states, the developers also have a complete blueprint of the behavior of the smart contract.

The two main things that need to be added to the PSM manually in order to finish the smart contract are contract variables and transition attributes. The contract variables are needed because Solidity is statically typed and every variable needs to be defined. If a timed transition is going to take place in the contract, a contract variable containing, for instance, the creation time needs to be defined. Another frequently seen contract variable is the Boolean operator, which denote if a condition is true or false. Contract variables can be discovered by looking at the set of transitions.

The transition attributes formalize the transitions between states. The transitions are done through functions in Solidity, for which the guard condition, input, statements, and output needs to be defined. The automatically generated fundament of the smart contract already denotes the state it comes from and to which state is goes. In the PDD in Figure 5, the process and deliverables of creating contract variables and transition attributes is shown. Hereafter the transformation from the FSM to Solidity is complete. The PDD containing the full method can be found in Appendix A.

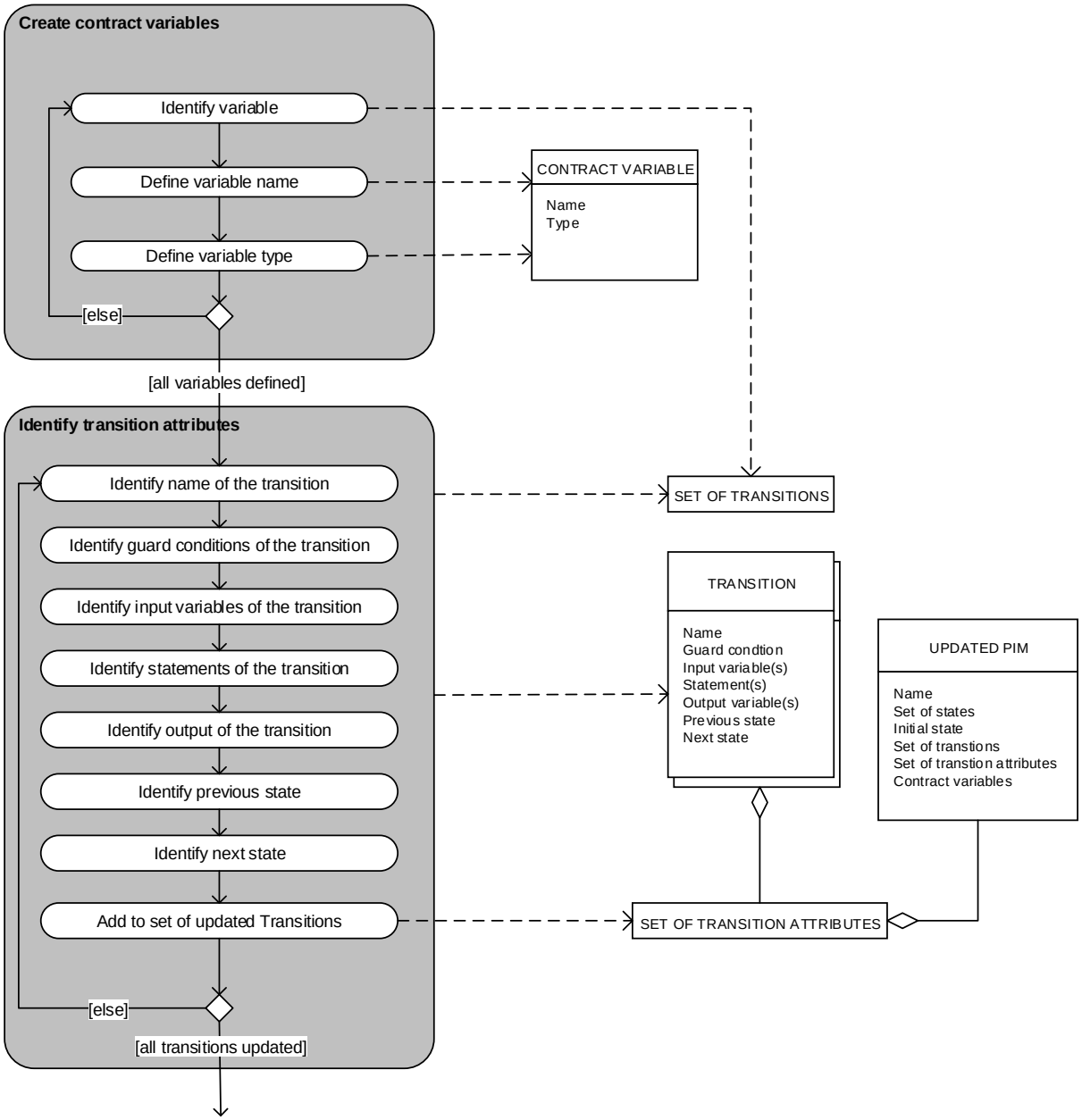


FIGURE 22: PDD CREATING THE PSM

5. Case Study Evaluation

In this chapter, the model-driven smart contract development framework is assessed by developing a smart contract, going step-by-step through the method. The findings of the case study will be discussed at the end of this chapter.

5.1 Case Study

The case chosen for this assessment lies in the domain of property leasing, which is an ideal industry for the application of blockchain technology and the usage of smart contracts. Property-related information is increasingly available in digital form, but a lot of the information is scattered among disparate systems. This leads to a lack of transparency and efficiency in finding and renting a living space. The application of blockchain could have an impact on this industry. It can facilitate a common database in which listings are collated in a central place. Multiple entities can access and modify a variety of information, and the lack of trust among entities can be resolved through blockchain properties. Classically, intermediaries are needed, which also become superfluous. Figure 23 gives an overview of why blockchain and smart contracts are a viable solution for the property leasing industry.

For property leasing to benefit optimally from the potential benefits of blockchain, the leasing contracts have to be reworked from paper contracts to smart contracts. The lease contract is one of the most common paper contracts, so this reworking has a big impact. For this reason, it is important that the development process is well thought out and the end product has no deficiencies relative to the paper contract.

The description of the case study is based on the contents of a paper contract tailored especially to a rental agreement between a tenant and a landlord, with a third party who performs maintenance and an initial and final check of the apartment. From this paper contract, the description of the case can be formed.

FIGURE 23: OVERVIEW OF ADDED VALUE OF BLOCKCHAIN IN THE PROPERTY LEASING INDUSTRY

This contract is between a landlord and a tenant. The landlord owns property, which will be leased to the tenant. The tenant pays the landlord a monthly rent, the amount of which is determined at the launch of the contract. The lease period is one year in which both parties cannot breach the agreement. After one year, the lease period can be extended, after which both parties can end the contract, provided that a one-month notice is given.

To safeguard against the tenant not paying the rent or otherwise breaching the contract, a security deposit of two times the rent is paid at the start of the contract. Also at the start of the contract, a third party assesses the state of the property. This also happens at the end of the contract, so that it can be determined if the tenant is liable for damages. If there are no damages detected, the tenant will recover its security deposit. If there are damages, these will be retracted from the security deposit.

As mentioned, the rent is due each month. The tenant has a five-day period in which this rent can be paid. If the tenant fails to pay within this period, he will be fined. He then has another five-day period to pay the rent with the additional fine. After this period, the landlord has the option of terminating the agreement and the tenant will not get his security deposit back.

From this description, the smart contract development process is started with the creation of the CIM. Firstly, the attributes of the contract are defined, shown in Table 8.

Attribute	Description
Landlord	The person who owns the property and rents it out
Tenant	The person who wants to lease the property
Property manager	The person who checks the state of the apartment

TABLE 8: THE ATTRIBUTES OF THE SMART CONTRACT

With these attributes and the description, the ADICO statements can be formulated. These ADICO statements can be norms, with the attribute, deontic, aim, and condition, or it can be rules, with the attribute, deontic, aim, and condition. The set of statements can be found in Table 9, grouped by attribute.

The next step in the method is the evaluation of the set of ADICO statements. In the formation of the set, the syntactical correctness has been a main focus, as was the inclusion of all the domain knowledge. All the statements contain at least an attribute, deontic, aim, and condition, so the syntactical correctness is guaranteed. The completeness of the set was assessed in collaboration with domain experts on the subject of commercial real estate and contractual law. This resulted in the addition of two statements. It would be redundant to show the entire table two times, so these statements are incorporated in Table 9. After this, the completeness of the set of ADICO statements was accepted.

Attribute (A)	**Deontic (D)**	**aIm (I)**	**Condition (c)**	**Or else (O)**
Tenant	must	pay the security deposit	at the start of the contract	
Tenant	must	pay the rent	every month	or else the rent is late
Tenant	must	pay the rent and a fine	if the rent is late	or else the contract can be terminated
Tenant	may	end the lease contract	when the original lease period ends	or else the lease contract may be extended
Tenant	may	give notice	before the lease period ends by forfeiting his security deposit	
Tenant	may	give one-month notice to leave through alternative pay rent option	if the original lease period has ended	
Landlord	must	launch the contract	at the start of the contract	
Landlord	may	terminate the contract	if the rent is ten days late	
Landlord	may	keep the security deposit	if the rent is ten days late	
Landlord	may	extend the lease contract	if the lease period ends	or else the lease contract will end
Landlord	may	end the lease contract	when original the lease period ends	or else the lease contract will be extended
Landlord	may	terminate the contract	if notice has been given and final rent is 10 days late	
Landlord	must	return the security deposit to tenant	if the end of the lease has been reached	
Property manager	must	asses state of the apartment	at the start of the contract	

Property manager	must	asses state of the apartment	at the end of the contract	
System	must	shift to accepting payments	every month	
System	must	register the rent to be late	if five days have passed since the accepting payment status has been reached	
System	must	end the lease contract	if final payment has been made within 12 months	

TABLE 9: ADICO STATEMENTS

From the set of ADICO statements, the transformation to the PIM can be made. The name of the PIM is "Property leasing finite state machine diagram." The initial state can be determined by seeing what happens when the contract is launched. It is determined that the Landlord launches the contract, so the initial state can be named "Created." All possible scenarios of this case will end in the lease contract between the Landlord and the Tenant being concluded. This state can be named "Finalized." The set of states can be elicited from the ADICO statements by looking to what happens before and after an ADICO statement. For instance, looking at the first statement "Tenant must pay the security deposit at the start of the contract or else the contract is not initiated", the first state is one where the deposit has not been paid, and the next state is one where the deposit has been paid, with the transition being the payment of the deposit. Using this method, a set of states and transitions can be formed. These sets are shown in Tables 10 and 11. There is an element of creativity to the creation of the sets, as there are multiple ways to model the same set of ADICO statements. An overload of states is undesirable, so if possible, states should be combined.

State name	**Description**
Created	This is the initial state. The Landlord has created the contract.
PropertyAssessed	When the Property manager has assessed the condition of the property, this state is reached.
Paid	This is state in which a payment has been made. This can be the security deposit, the rent, or the late rent with the additional fine.
AwaitingPay	In this state, the monthly rent is due, which means it is reached every month.
Conflict	If the Tenant fails to pay the rent within the initial period, this state is reached. It marks that there is a conflict.

PaidExtended	This state resembles the state Paid, with the addition that a 12-month period has passed. This means that the Tenant is now able to give a one-month notice without losing his security deposit. This will be possible if the AwaitingPayExtended state has been reached.
AwaitingPay-Extended	This state resembles the AwaitingPay state, with the addition that a 12-month period has passed. This means that the Tenant is now able to give a one-month notice without losing his security deposit. From this state, notice can be given.
ConflictExtended	This state resembles the Conflict state, with the addition that a 12-month period has passed.
FinalToBePaid	This state is reached when a party gives notice that the agreement will be ending. This can be done by the Tenant or by the Landlord. This state marks that there is going to be one last payment to do.
LeaseEnd	This state is reached when the final payment has been made by the Tenant.
LeaseClosed	If the Tenant fails to pay his last payment within the designated time period, this state can be reached by paying the late rent plus the fine.
Finished	This is the final state of the contract. Every possible scenario should lead to this final state. When reached, the contract is finished and could be deleted.

TABLE 10: STATE DESCRIPTION

Transition Name	**Description/ADICO statement it is based on**
Create	When the Landlord launches the contract, this transition is made. It saves the contract to the blockchain.
	Landlord must launch the contract at the start of the contract
Assess	In this transition, the Property manager inspects the condition of the property at the start of the contract.
	Property manager must assess state of the contract at the start of the contract.
PayDeposit	The Tenant pays the security deposit.
	Tenant must pay the security deposit at the start of the contract.
RentDue	This transition is an automatic one done by the system. When 30 days have passed, the system automatically changes state from Paid to AwaitingPay.
	System must shift to accepting payments every month
PayRent	The Tenant is the trigger for this transition, in which he transfers currency to pay for the rent, transitioning the state to Paid.
	Tenant must pay the rent every month or else the rent is late.

RentLate	If the Tenant fails to pay the rent within the five-day time period, the system will automatically shift to the next state: Conflict. The condition for this transition is that five days have passed since the state AwaitingPay has been reached.
	System must register the rent to be late if five days have passed since the accepting payment status has been reached.
PayLate	This transition is triggered by the Tenant, who pays the rent plus a fine for being late with the rent. If the contract is in the extended period and notice has been given, this transition leads to LeaseClosed.
	Tenant must pay the rent and a fine if the rent is late or else the contract will be terminated.
EndLease	If the contract has run for a period of 12 months, the Tenant or the Landlord can end the lease. If the final payment has been made within the 12 month period, the system automatically shifts to the finished state.
	The Tenant may end the lease contract when the original lease period ends or else the lease may be extended. Landlord may end the lease contract when the original lease period ends or else the lease contract will be extended. System must end the lease if final payment has been made within 12 months.
EarlyTerminate	The Tenant may give notice before the contract period of 12 months is over, but he forfeits his security deposit by doing so. The condition for this transition is that the lease period is under 11 months.
	Tenant may give notice before the lease period ends by forfeiting his security deposit.
Extend	When the original lease period of twelve months has passed, the Landlord may extend the lease period by this transition.
	Landlord may extend the lease contract if the lease period ends or else the contract will end.
GiveNotice	When the lease is extended and the rent is due, the Tenant may give notice. This is done by paying the rent via this transition.
	Tenant may give one-month notice to leave through alternative payment if the original lease period has ended.
TerminateContract	If the contract is in the conflict state and the Tenant fails to pay within a five-day period, the Landlord may choose to terminate the contract. This means the contract ends and the Tenant will not be refunded his security deposit.
	Landlord may terminate the contract if the rent is ten days late. Landlord may keep the security deposit if the rent is ten days late. Landlord may terminate the contract if notice has been given and final rent is 10 days late.
ReturnDeposit	When the final payment has been made by the Tenant, the Property manager inspects the state of the property again. If the property is in a similar state as in the first assessment, the Tenant will receive its security deposit back.
	Property manager must assess the state of the apartment System must return security deposit if the assessment of the apartment is similar to the first assessment.
DepositRetract	The Property manager assesses the property and compares it to the first assessment. If there are damages or other expenses caused by the Tenant,

	these will be retracted from the security deposit. After this, the remainder of the security deposit is returned to the Tenant.
	Property manager must assess the state of the apartment. System must register the amount and retract from security deposit if the assessment has been made.

TABLE 11: TRANSITION TABLE

With the set of states and the set of transitions, an FSM can be constructed. It is desirable for this FSM to have as little overlapping transitions as possible. This results in the FSM in Figure 24; an enlarged version can be found in Appendix B. Again, the correctness and completeness are assessed. The correctness of the model is dependent on a number of factors. Every state should have a transition, which it has. The model does not contain unwanted loops, and every scenario leads to the final state. Where it is applicable, the condition is shown in brackets underneath the transition name. All the states from the set of states and all the transitions from the set of transitions are modeled. Every statement from the set of ADICO statements is modeled, which means that the all the essential behavior is modelled. Thus the model is complete.

With this model, the next step of the MSDM can be initiated. This is extending the model in order to prepare it for the transformation to the PSM. The easiest way is go through the ADICO statements and look for nouns which do not describe the attributes. The contract variables and their types are shown in Table 12.

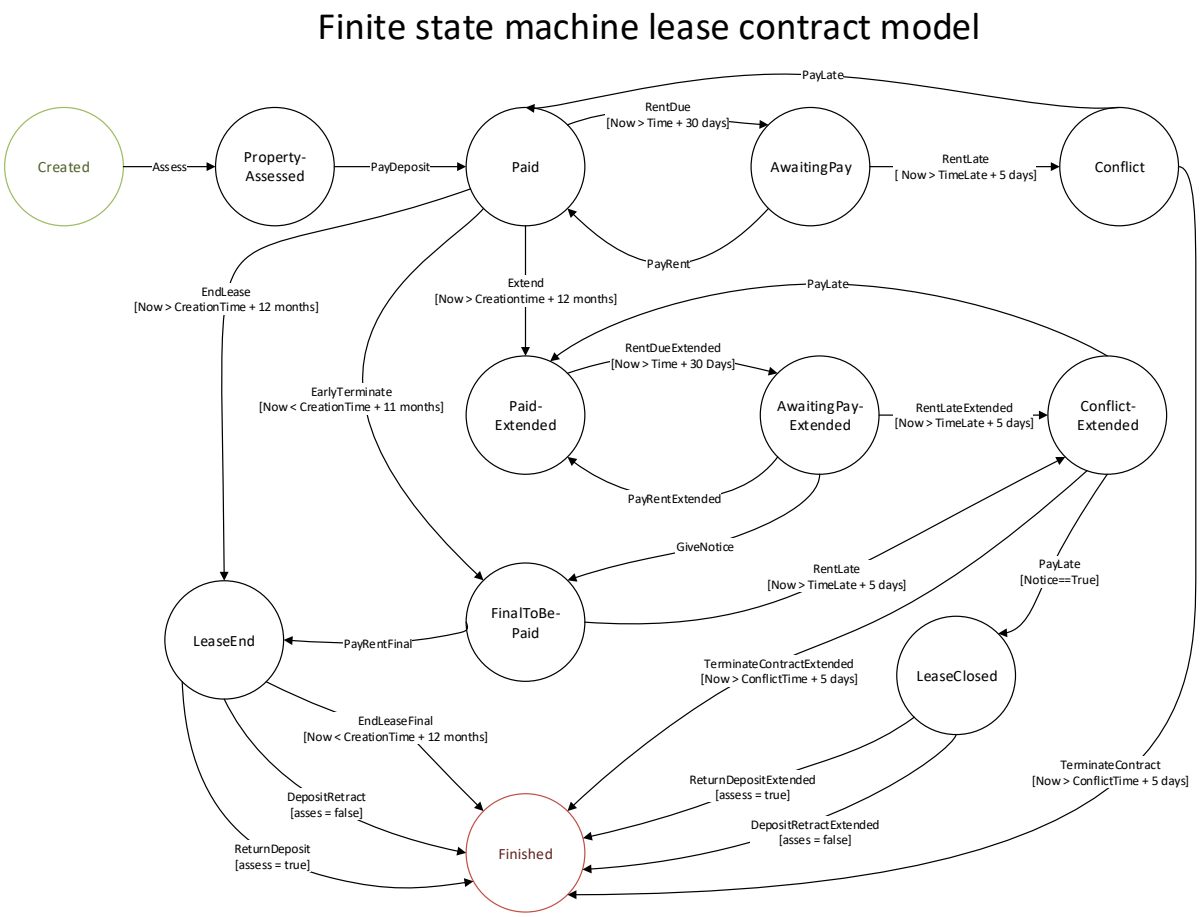


FIGURE 24: FSM LEASE CONTRACT MODEL

Variable name	**Variable type**
Security deposit	Uint
Rent	Uint
Fine	Uint
CreationTime	Uint
Landlord	Address
Tenant	Address
PropertyManager	Address
PayDeposit	Bool
Notice	Bool
State of the apartment	Enum

TABLE 12: VARIABLE DEFINITION

The next step is the actual creation of the smart contract. The creation of the contract can be divided in four main parts, namely the definition of the states, the definition of the variables, the definition of the patterns used, and the coding of the transitions. First, the states are declared, which is done using the states from the FSM. The declaration of the states is fairly easy, taking all the states from the set of `states`, and defining them in an enum. Under the declaration of the states, the initial state is declared. The StatesDefinition is as follows:

```
	//StatesDefinition
	enum States {Created, PropertyAssessed, Paid, AwaitingPay, Conflict,
PaidExtended, AwaitingPayExtended, ConflictExtended, FinalToBePaid,
LeaseEnd, LeaseClosed, Finished}
	States public state = States.Created;
```

After this, the variables are declared, with their type, an access modifier, and a name. The variables definition is as follows:

```
	uint public securityDeposit;
	uint public rent;
	uint public fine;
	uint public deposit;
	uint monthCounter;
	uint rentCounter;
	uint creationTime;

	bool private payDepositTrue;
	bool private notice;
	bool assess;

	address public landlord;
	address public tenant;
	address public propertymanager;
```

Then, for every transition, a function is made containing a guard in the form of the state it comes from, and a declaration of what the next state is. Also, for every function is determined if one or multiple patterns are applied to the function. After this, the statements and additional guard condition have to be added manually, which does require a basic level understanding of the Solidity programming language, but this can be expected of a smart contract developer. By providing the skeleton for the contract, the risk of completely altering the smart contract in an undesirable manner is mitigated. The result of the case study in the form of a smart contract skeleton is shown in Appendix C.

5.2 Case Study Findings

The application of the model-driven smart contract development method to a case has demonstrated the activities and deliverables of the method. Furthermore, the instantiation of the method has yielded various findings and discussion points. The results are discussed per sub-activity, after which the findings and discussion points on the method as a whole are discussed.

5.2.1 The Computational Independent Method

Finding: The CIM provides a blueprint for the development process by structuring the requirements and provides a framework for the evaluation of the smart contract.

The grammar of institutions used in the CIM allows the participants in the development process to structure the requirements and make the interactions actors have with the smart contract explicit. This provides a schematic which forms a basis for the remaining development phases. Both the PIM, PSM, and the code should correctly reflect the ADICO statements. At every step of the process, the set of ADICO statements can be consulted to ensure the requirements of the smart contract are fulfilled. As the statements must not be contradictory, the instantiation can, in theory, also never be contradictory. This results in smart contracts without possible paradoxical behavior.

5.2.2 The Platform Independent Model

Finding: Creating the PIM can be time-costly and laborious.

The process of creating and evaluating the PIM took up more time than initially thought. The FSM modelling technique is straightforward with states and transitions between these states. However, in this case it became clear that if not modelled with caution, the number of states and transitions can easily become very large. This way the FSM may become cluttered and hard to read.

Finding: The FSM modelling technique is suitable for the incremental construction of models.

The FSM allows for an incremental construction of the model. In this case study, the interactions between the landlord and the tenant were modelled first, after which the property manager was added to the model. Consequently, if other parties or new clauses were to be added to the smart contract, these can be added to the existing model with relative ease.

Finding: The PIM can be used to evaluate the possible flows through the smart contract.

When the FSM model is finished, it can be used to evaluate the smart contract. If constructed correctly, the FSM gives an easily comprehensible overview of the smart contract and allows for the evaluation of all the possible flows to the contract. Combined with the CIM, developers can check if all the needed requirements are modelled and if there are no unintended flows through the contract are possible.

5.2.3 The Platform Specific Model

Finding: Constructing the smart contract as a FSM makes the smart contract less vulnerable, but also has a downside.

By applying a state machine approach, the number of functions that can be accessed and executed are drastically lowered. An advantage to this is that when the tenant needs to pay his rent, no function can be activated to, for instance, drain the security deposit. The state machine approach does come with a downside. There can be an explosion of functions when the number of states increases. The contract in the case

study is not extremely complex, but already has a high number of functions to connect all the states.

Finding: The patterns in combination with the state machine approach mitigate a large part of the smart contract vulnerabilities.

As stated in the previous finding, the state machine approach mitigates vulnerabilities by reducing the number of functions that can be executed. The described patterns amplify this effect. The application of the patterns is not time-costly, because they are declared as modifiers once and then easily applied throughout the smart contract.

Finding: The manual coding process is relatively easy, but can be time-costly.

This finding logically follows the downside of approaching the smart contract as a state machine. Every transition between states needs to be manually coded as a function. These functions are not complex because only the conditions for the transitions need to be added, but with an increasingly large amount of transitions this can become time-costly.

5.2.4 Overall Findings

Finding: The method facilitates a structured, formalized approach to smart contract development.

This is a logical finding, but the case study has shown that the method has clear activities and deliverables, which form a guidance in the creation of a smart contract. The method can be followed linearly, but as it was found in this case study, sub-activities can be revisited with relative ease in order to make the method a more incremental approach.

Finding: The method can be time-costly.

As stated in the finding on the PIM, the creation of the models can be a time-costly process. Based on this case study, the conclusion that a model-driven approach to smart contract development is more efficient is a bit premature. However, as the method is time-costly, it does yield the CIM and the PIM, which can both be used to evaluate the completeness of the smart contract. Furthermore, these deliverables provide a basis for the re-use of the business knowledge.

6. Experimental Evaluation

In this chapter, the model-driven smart contract development method is evaluated by conducting an experiment aimed at evaluating the added value of the CIM to the method. The goal-question-metric approach is used to define the metrics and the execution and results are described in detail.

6.1 Goals, Hypotheses, and Variables

Embedding metrics into a goal-oriented framework is widely regarded as a good practice [99]. The Goal-Question-Metric (GQM) approach is such a framework. It is a top-down approach developed by Basili [100] and later extended by Rombach [101]. Top-down means that first specific goals are stated, then questions whose answers will help attain the goals are asked. Lastly, the metrics are defined to provide a scheme for measuring.

Goals in the GQM are preferably documented in a structured way using the following template: Goals are defined for a purpose (e.g., understanding, improving, controlling), for the object under study (e.g., process, product), from a perspective (e.g., user, developer), and within certain context characteristics (e.g., involved persons, environment, resources, organisations). The addition of a CIM to the MDA framework is to better communication and understanding between people in the development process, about the models with a higher computational rich load.

Following the template for goal definition that is suggested in [102], the goal of this experiment can be summarized as follows:

Analyze the Finite State Machine modelling technique
For the purpose of evaluation
With respect to comprehension and efficiency
From the point of view of the researcher
In the context of participants comprehending FSM models

From this goal, the main goal of the experiment at hand can be distilled. The goal of the experiment is to evaluate if the comprehension of Finite State Machine models is influenced by the addition of a CIM from the viewpoint of the users who do

not have a development background. After the goal is determined, questions are needed to instantiate the goal of the experiment. These questions help attain the goal and are formulated as follows:

Q1: *Which technique leads to the highest comprehension of the Finite State Machine modelling language?*

Q2: *Which technique is most efficient in understanding the Finite State Machine modelling language?*

With the goal and the questions defined, the last step in the GQM is determining what metrics are best suited for answering the questions. The metrics for this experiment are the comprehension score, the comprehension score broken down in categories, and the time. The experiment is summarized in Table 13..

Variable	Question	Metric(s)
Comprehension	Which technique leads to the highest comprehension of the FSM?	Comprehension score
		Categorized comprehension scores
Efficiency	Which technique is the most efficient in understanding the FSM	Time

TABLE 13: VARIABLE, QUESTION, METRIC OVERVIEW FOR THE EXPERIMENT

6.2 Hypotheses

Using the goal, the questions and the metrics, hypotheses can be formulated. The hypotheses are defined by following the guidance from the literature, which states that the CIM can increase comprehension and provide a framework for the communication of the PIM. They are formed using the h_0 h_1 style.

Hypothesis 1_0 : There is no significant difference between subjects comprehending an FSM model with a textual description and subjects comprehending an FSM model with a CIM with respect to their comprehension score.

Hypothesis 1_1 : There is a significant difference between subjects comprehending an FSM model with a textual description and subjects comprehending an FSM model with a CIM with respect to their comprehension score.

Hypothesis 2_0 : There is no significant difference between subjects comprehending an FSM model with a textual description and subjects comprehending an FSM model with a CIM with respect to their efficiency.

Hypothesis 2_1 : There is a significant difference between subjects comprehending an FSM model with a textual description and subjects comprehending an FSM model with a CIM with respect to their efficiency.

6.3 Design

The design used for this experiment is the static group comparison. The static group comparison is a design in which a group who has experienced a treatment is compared to a group who has not experienced said treatment to establish the effect of the treatment [103]. The treatment in this experiment is the addition of a CIM to assist in comprehending the PIM.

Giannatasio states that the most important threats to a static-group comparison design are the selection bias and the interaction between subjects [104]. The threat of selection bias means that when two groups are formed, one group has an advantage over the other group, skewing the results. The selection bias was mitigated by randomly dividing the participants among the two groups, with the assumption that all participants had equal knowledge at the start of the experiment. The threat of interaction between subjects means that participants that have already partaken in the experiment communicate the workings to participants who have not, giving those participants an advantage over the other participants. The subjects were requested not to talk about the experiment until all participants had finished the experiment in order to mitigate the threat of interaction between participants.

The experiment is executed in a single instance, so there is no pre-test. This approach is chosen to mitigate the effect of testing on the experiment. Testing and retesting often influences participants behavior [105]. As smart contracts are a relatively novel concept, the participants might encounter the concept in this form for the first time at the experiment. Testing after a period of time might influence how they perceive the smart contract, which is an unwanted effect on the experiment. A commentary on this design is that there is no baseline to compare the influence of the treatment to. However, by selecting and randomizing a heterogeneous group of

participants (elaborated on in the next section), one might assume that the treatment is the cause of a potential difference in the results.

6.4 Subjects

This thesis is written during an internship project at Deloitte Risk Advisory. The participants were all employees of this department. This provides a satisfying level of assurance that the measured outcome is an effect of the treatment and not an effect of the heterogeneity of the participant pool and differences between the two experimental groups. All the subjects have had similar education, have affinity with IT, and work with information systems on a daily basis. However, none of them have experience with developing smart contracts, which was checked before the experiment in order to rule out unfair advantages due to preexisting knowledge. The level of experience in conceptual modeling and modeling language was also registered. In order for the groups to be homogenous, the prior knowledge should have no effect on the comprehension score. This will be analyzed in the results. There was a total of 16 participants in this experiment, equally divided among the two subject groups.

6.5 Instrumentation and Procedure

Each participant in either group received an introductory coversheet which stated all the materials needed for the experiment and described the tasks to be executed. After a check was done if all the materials were present, the participant was asked to start reading. The materials for this experiment can be found in Appendix D. The reading part of the experiment comprised the following components for both versions:

- **An introduction to finite state machines**, in which the FSM modeling technique is explained;
- **A description of the lease contract**, in which the case for the smart contract is explained;
- **A finite state machine model** based on the case description.

The version for the treatment group had the following additional components:

- **An introduction to ADICO**, which is a description of the CIM modeling method;
- **ADICO statements**, which is a CIM which complements the finite state machine model (PIM) provided.

There was no designated reading time, allowing the participants to read at their own pace. To see if the participants had read and understood the materials, a cloze test for comprehension was used. This is a test with statements in which concepts have been blanked out. It was not used for further evaluation as the difficulty was low, but it served as a comprehension check and ensured that the participant had a sufficient understanding of all the materials presented to them.

After a participant had finished reading and had shown a sufficient understanding of the concepts, the comprehension task started. Two scenarios of the finite state machine model were shown to the participant. These scenarios were marked with a red line in the FSM model the participants had already received. The participant is then asked to describe the steps shown in the scenario in as much detail as they can. Participants were explicitly asked to write down their answers as extensive as they can, leaving no knowledge implicit. This was done to ensure that their full comprehension becomes visible in their answer. The model in experiment B was accompanied by the ADICO statements, with the number of the statements corresponding to the number shown above the transition names. It was verbally confirmed that the participants understood this connection so that the participants would not ignore the CIM while describing the scenarios.

The FSM model was a variation on the FSM created in the use case in chapter 5. Two parties, namely the landlord and the tenant, have a rental agreement denoted in the smart contract. In the comprehension task, both groups were shown the same two scenarios in which the flow through the FSM was indicated by a red marking. The FSM in the scenarios was the same, only the marked scenarios differed. The first scenario showed a desirable flow through the FSM in which the actors did not trigger conditions which warranted punishment or fines. The second scenario had a flow through the model in which unwanted behavior such as not paying on time was

displayed. By having two flows through the FSM that show different behavior through the model, the comprehension of the complete FSM has been attempted to be measured. The scenarios used in the experiment can be found in Appendix D2 and D4.

The participant's answers were scored using a control sheet in which components were all scored with one point. The control sheet used can be found in Appendix E. For every participant, the points were summed and divided by the total points, creating the comprehension score. This score is based on the recall, in which the relevant items are divided by the total number of items [106]. The recall is eligible in this context, because the participants' description were comprised of true positives and rarely false positives, or negatives. The time that the participants took to finish the scenario description was also measured. This was done in order to get an insight in the efficiency of the participants.

The experiment was performed in the same setting for each of the participants. The reading parts were provided in print and the FSM models were displayed on a 22 inch monitor on which all the components of the models were clearly visible. This way the participants were able to consult the information provided, while simultaneously comprehending the FSM models. A separate laptop was provided on which the participants could type their answers. All participants were able to do the experiment in a calm setting without getting disturbed while the experiment was in progress.

In addition to the quantitative measurement of the comprehension score and the efficiency, a qualitative evaluation of the participant's abilities was made. This is done because comprehension is a hard metric to measure by just looking at the numbers. By doing an additional observation, an explanation can be formed about the participants' scores and can be assessed if the measurement instrument used for this experiment is reliable. The data was collected through post-experiment interviews in which the researcher sat down with the participant and verbally discussed their answers to the comprehension tasks. When the participant had left out components or had made a mistake, the reason for this was sought after in order to ensure that the measured comprehension score was a valid representation of the participants' actual comprehension.

6.6 Data Collection and Results

6.6.1 Quantitative Data

All analyses described in this chapter were done in IBM SPSS statistics version 24, the output of the analyses can be found in Appendix F. The first analysis that was conducted was the effect of participants' prior knowledge on the comprehension score. This has been evaluated through an independent samples T-test and showed no significant result (p = .885 for scenario 1 and p = .777 for scenario 2). This means that having prior knowledge on conceptual modelling did not have an effect on the participants' comprehension score, so the assumption can be made that all participants were equally capable.

Hereafter, the comprehension score was analyzed. The comprehension score is the result of scoring the participants' description of the scenarios against the control sheet. The participants could acquire 42 points for 42 components they could have mentioned in the scenario description. The comprehension scores calculated as percentages for scenario 1 and 2 are displayed in figure 25.

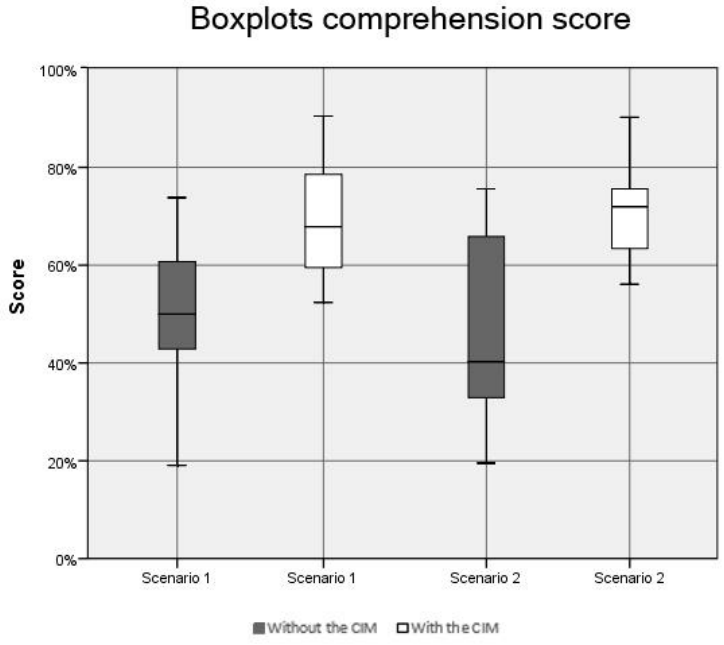


FIGURE 25: COMPARISON OF THE MEAN SCORES IN SCENARIO 1

In both scenarios, the group with the CIM scored higher. To see if the difference in scores was a significant effect of the condition, two independent sample T-test were conducted in which the scores of the group without the CIM were compared to the

scores with the CIM. For scenario 1, there was a significant difference between the comprehension score of the group without the CIM (M=50, SD=16.6) and the comprehension score of the group with the CIM (M=69.3, SD=12.6); t(14) = -2.629, p = 0.020, N = 16. For scenario 2, the T-test also shows that there was a significant difference between the comprehension score of the group without the CIM (M=46.6, SD=20.1) and the group that used the CIM (M=71.0, SD=10.5); t(14) = -3.041, p = 0.009, N = 16. This result suggests that people are better able to comprehend an FSM when using a CIM than when they are not using a CIM. This means that for both scenarios, the null hypothesis of hypothesis one is rejected, and the alternative hypothesis is accepted.

Hereafter, the comprehension score was divided in five categories, namely the actor, action, construct, consequence, and condition category. The participants' scores for each of these categories were then calculated as a percentage of the total possible scores. The results for the scores in scenario 1 visualized as boxplots are shown in Figure 26.

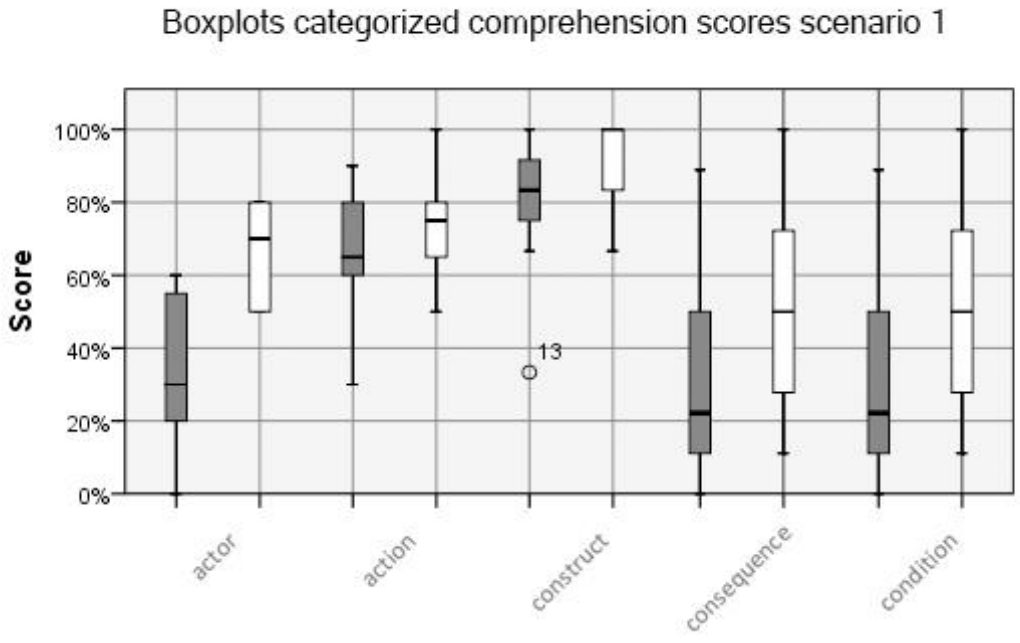


FIGURE 26: BOXPLOTS OF THE CATEGORIZED COMPREHENSION SCORES FOR SCENARIO 1

As one can tell from the boxplots, the group with the CIM scores, on average, higher on all the different components. Five independent sample T-tests showed that of the

five components, only the actor component showed a significant difference between the group without the CIM (M=33.8, SD=21.3) and the group with the CIM (M=66.3, SD=18.5); t(14) = -3.596, p = 0.003, N = 16. This suggests that people interpreting an FSM with a CIM are better able to denote which actor is responsible for an action than people without a CIM. The other components showed no significant difference between the two groups.

The same was done for the comprehension score of scenario 2. The results of these categorized comprehension scores can be found in Figure 27.

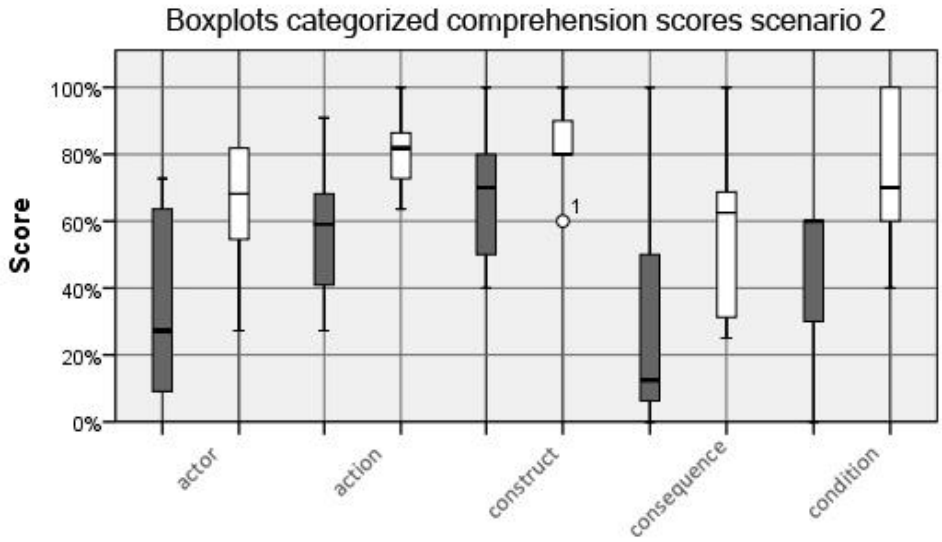


FIGURE 27: COMPARISON OF THE MEAN SCORES IN SCENARIO 2

The second scenario shows the same trend as in scenario 1. The group that comprehended the FSM with the CIM scores relatively better on average on the total score, as well as on all the component scores. Again, five independent samples T-tests were conducted to establish if the effect was significant or not. For the five components in the second scenario, three effects were found. There was a significant difference in the actor score between participants without the CIM (M=35.2, SD 28.1) and participants with the CIM (M=64.8, SD=19.7); t(14) = -2.435, p = 0.029, N = 16. There was a significant difference in the action score between participants without the CIM (M=56.8, SD=20.5) and participants with the CIM (M=80.7, SD=11.3); t(14) = -2.885, p = 0.012, N = 16. And there was a significant difference between the condition score between participants without the CIM (M=45.0, SD=23.3) and participants with the CIM (M=75.0, SD=23.3); t(14) = -2.575, p = 0.022, N = 16. These results suggest that

people interpreting an FSM with a CIM are better able to comprehend the actors, the actions, and the conditions of the FSM than people who do not utilize a CIM. For the construct and consequence components, there was no significant result found.

After the analyses of the comprehension scores, the efficiency of both groups was analyzed. An independent samples T-test showed that there was no significant difference in time used between the group comprehending with the CIM and the group comprehending without the CIM. This means that the treatment had no effect on the efficiency, so the null hypothesis of hypotheses 2 is accepted. Additionally, a Pearson's correlation test was used to analyze if time had an effect on the comprehension score. This showed a moderate correlation between the time in seconds used for comprehending scenario 1 and the comprehension score of scenario 1, but this correlation was not significant ($r = .453$, $p = .078$, $N = 16$). A Pearson's correlation test showed a similar correlation between the time in seconds used for comprehending scenario 2 and the comprehension score of scenario 2, but this also was no significant result ($r = .436$, $p = .094$, $N = 16$). For both scenarios, there was a moderate positive correlation between time and comprehension score meaning that if participants took longer, they scored higher. However, both of these results were not significant. A possible reason for this is the relative low number of participants. This does mean the results do not yield any real conclusions.

6.6.2 Qualitative Data

With the knowledge gathered from the quantitative data of this experiment, the qualitative data can provide further insights into the results. The answers of the participants were reviewed in which the participants were asked about missing components in their answers. From these small interviews, it became apparent that there were two main reasons that participants left out information in their answers, which became most apparent for the actor category.

The first reason was that participants left certain information implicit with the expectation that it would be redundant to include them in their answer. An example in which this frequently happened was naming the actor who paid the rent. A lot of participants who left this out indicated that they did not name this actor because they assumed it was evident that the tenant paid the rent. "*Who else would pay the rent?*".

The second reason was that participants simply did not knew who the actor was. If the measurement instrument was reliable, the second reason would be the explanation for almost all the missing information in the participants' answer, correctly reflecting the participants' understanding. Unfortunately, the qualitative data shows that this is not the case.

However, there was a difference visible between the two scenarios. The first reason, namely participants leaving their comprehension implicit, was more frequent in the first scenario in which the tenant paying the rent is a recurring transition. The second reason, namely not knowing the answer, was more frequent in the second scenario in which some transitions were wrongly interpreted to be executed by another actor. The results of the second scenario therefore can be seen as more reliable in the effects it showed, compared to the effects displayed by the results of the first scenario.

As the FSM has no explicit information on who performs the action, this should be derived solely from context. In the scenario in which the contract was working out as it should, this did not seem to be a problem, and all participants could show who triggered which transition. In scenario 2 however, in which undesirable behavior for the contract is displayed, the agency was less clear to the participants without the CIM.

In the results above, the agent category was taken as the prime example of the discrepancy between the results of the quantitative results and the qualitative results. However, there were other examples in which the participants left out components of an answer for another reason than that they did not comprehend the correctly. As the categories are a subset of the comprehension score, the effects measured on these results are subject to the same threat. As the second scenario was less prone to a discrepancy between the quantitative and the qualitative results, there still is a level of validity to the described effects. However, when discussing these effects, it should be noted that the measurement instrument was not 100% reliable.

7. Conclusions

This thesis has presented a model-driven smart contract development approach in which the requirements of a smart contract are stepwise developed into a skeleton Solidity smart contract. This development method is an attempt to answer the main research question.

RQ: *How can smart contract development be supported by model-driven engineering in a structured way?*

This main research question has been deconstructed into smaller parts, which all provide valuable insights into model-driven smart contract development. These obtained insights will be presented in this chapter. To arrive at an answer to the main research question, the sub-questions will be treated and answered shortly. Most questions have been answered more elaborately in a chapter of this thesis, so the relevant chapters for a more extensive answer will be given.

SQ1: *How can blockchain, smart contracts, and model-driven engineering be defined based on prior literature?*

The aim of this first sub-question is to provide an insight into the concepts blockchain, smart contracts, and MDE. Blockchain can be summarized as an immutable, append-only, decentralized database. The blockchain technology is a combination of cryptography, peer-to-peer networking, and consensus protocols. Its properties can facilitate major innovations, but do not come without vulnerabilities. The same goes for the smart contract, which can be defined as code that is stored and executed on the blockchain. The idea of a smart contract stems from 1997, but blockchain technology has provided a platform for the actual execution of these applications. Again, this novel technology provides opportunities but also has novel pitfalls which should be accounted for in the usage of this technology. A more comprehensive definition of blockchain and smart contracts is given in chapters 3.1 and 3.2.

MDE is a more established concept that has been applied in many fields. In this methodology for software engineering, or engineering in general, the focus shifts from a code-centric to a model-centric approach to development. Possible advantages of this approach are lowering the difficulty threshold and the re-use of knowledge in

volatile development environments. There are a number of frameworks and methods for MDE. MDA is such a framework, which provides a model-driven foundation for the development process. The MDA framework consists of the computational independent model, the platform independent model, and the platform specific model. These different models abstract from a systems' implementation by providing different levels of behavioral, functional, and technical detail. These concepts are described in more detail in chapter 3.3.

SQ2: *What research into the application of model-driven engineering to smart contract development has already been conducted?*

This sub-question combines the concepts defined in the first sub-question. Three main approaches to the MDD of smart contracts have been identified in literature. The agent-based approach in which the behavior of a system is modeled with the grammar of institutions [80], the process-based approach in which BPMN is used to create a basis for a smart contract between trustless parties [85], and the state machine approach in which the smart contract is modeled as an FSM [88]. These approaches are extensively described and compared in chapter 3.5.

SQ3: *What are the requirements for the model-driven smart contract development method?*

For this sub-question, the knowledge gathered in sub-questions 1 and 2 was used. The aim was to elicit the requirements based on the definitions made in sub-question 1 and to tailor these requirements to the available approaches already described in literature. As the MDA framework was applied, a distinction was made between the CIM, the PIM, and the PSM. The PSM in the context of this research was already decided to be the Solidity programming language, as this is by far the most widely used language for smart contracts. The full requirements are discussed in chapter 4.1, but the requirements can be summarized as follows:

CIM: The CIM should provide high-level overview of the smart contract, which maps the essential domain concepts in a systematic, structured manner and shows the developer all possible interactions with the smart contract.

PIM: The PIM should provide a functional and behavioral overview of the smart contract in which the notion that a smart contract is comprised of states is evident. The model should be extensive enough for the developer to create a functional overview, and accessible enough for the domain expert to communicate about this functional overview.

SQ4: *What are the activities and deliverables of the model-driven smart contract development method?*

The requirements denoted in sub-questions 3 form the requirements for this sub-question. Based on the approaches described in sub-question 2, a method base was created, and from this method base, the fragments best fulfilling the requirements were selected. For the CIM, the agent-based approach with the grammar of institutions was selected, because of the structured approach to describing the domain knowledge, while still being approachable to less technical people. For the PIM, the state machine approach was selected, because of its clear syntactical elements, its clear semantics, and the possibility to transform the FSM into a smart contract skeleton. The activities and deliverables of the smart contract development method are described in detail in chapter 4 but can be summarized as shown in Figure 27.

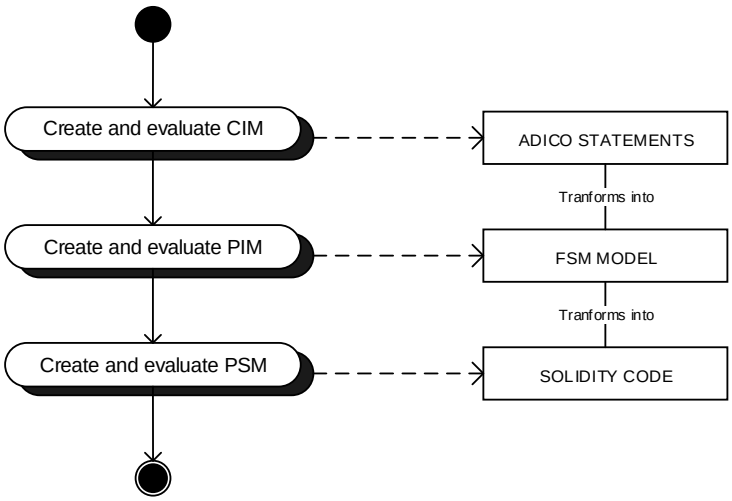


FIGURE 28: PDD SUMMARY OF THE MODEL-DRIVEN SMART CONTRACT DEVELOPMENT METHOD

SQ5: *How does the model-driven smart contract development method assist the developer in the creation of smart contracts?*

The first four sub-questions have resulted in the creation of the model-driven smart contract development method. Sub-question five and six are aimed at evaluating this method. The first evaluation is from the viewpoint of the developer and was done by assessing the method through a case study. This case study yields a number of conclusions.

For starters, the case study shows that the method provides the developer of the smart contract with a structured approach. By doing so, it is clear which steps need to be followed in order to create a smart contract. Not having a structured approach to development can lead to requirements not being fulfilled and can impede the preservation of business knowledge. The method also provides a structured approach to the translation of domain knowledge to smart contract requirements. This way, the developer does not have to rely entirely on his or her own mental representation of the domain knowledge, but is provided with a checklist of requirements instead.

Furthermore, the model-driven smart contract development method may assist the developer in creating less vulnerable smart contracts. First, the FSM is transformed into a skeleton contract in which the notion of a smart contract as a state machine is used. Having the smart contract be in a single state at a time mitigates the risk of having other functionalities of that smart contract tempered with at that moment. Furthermore, patterns can be applied which counter the reentrancy vulnerability and transaction-ordering dependency vulnerability, allow a form of access control and can assist in a timed transition. By providing a skeleton smart contract, the manual programming is kept at a minimum, and this programming is aimed solely on translating the transitions into code.

The case study also showed that the method can become time-costly. The construction and evaluation of the models can be a laborious process which increases with the complexity of the smart contract. However, the time invested does yield artifacts like the CIM and the PIM which can be used to asses the quality of the smart contract. An overview of all the findings from the case study can be found in chapter 5.2.

SQ6: *How does the model-driven smart contract development method influence the comprehension of smart contracts?*

The second evaluation of the method was done by conducting an experiment. The results of the qualitative analysis show that the measurement instrument might not be fully reliable and the results may be skewed by the participants own interpretations on what can be left implicit and what not. Therefore, the conclusions to this sub-questions need to be read with the limitations of the experiment in mind.

The main conclusion that can be drawn from this experiment is that the comprehension of FSM is higher when a CIM is provided. This legitimizes the addition of the CIM to the model-driven smart contract method as a comprehension tool for the domain expert. It is important for people with domain knowledge to be included in the smart contract development process and the addition of the CIM assists in this inclusion.

The results of the experiment show that people are better able to interpret concepts that are left implicit in the syntax of the FSM. The actor who triggers the transition, for instance, is better recognized as a CIM is provided. When a scenario gets more complex, the inclusion of a CIM also significantly betters the comprehension in terms of actions and conditions for those actions. This shows that the CIM is not only a tool to understand the FSM, but can also be used to make implicit information explicit.

More general conclusions from this experiment are that the FSM modeling language does not have a very steep learning curve, as the participants were all able to understand the FSM with a relatively simple explanation. However, in order to effectively communicate about what happens in the FSM, it helps to have a CIM containing information about the behavior of the system. Most participants were able to understand what happens in an FSM but were struggling to put this in writing when asked to. In these cases, the CIM can provide a framework to communicate what is happening in the FSM. Time and prior knowledge did not turn out to have an effect on the comprehension of FSM models. However, a non-significant positive correlation does suggest that when participants use more time to comprehend, the comprehension score increases.

RQ: *How can smart contract development be supported by model-driven engineering in a structured way?*

The answer to the main research question comes in the form of the model-driven smart contract development method presented in this research. The application of the MDA framework in the smart contract development environment (i) lowers the threshold of participation for domain experts in the development cycle, (ii) provides a structured manner for the developer to develop a smart contract based on domain knowledge, and (iii) lowers the chance of manual programming mistakes and smart contract vulnerabilities by providing the developer with a skeleton smart contract and patterns which can be used to mitigate smart contract vulnerabilities.

The components of the model-driven smart contract development method holistically form a structured method in which each of the components serves a purpose. The CIM forms a bridge between the domain knowledge and requirements for the smart contract, and it improves the comprehension of and communication about the FSM. The PIM captures the requirements and includes the functional aspects of the smart contract, which can be used as a blueprint for the PSM. And lastly, the PSM is the actual implementation of the smart contracts in which the state machine approach and the patterns are applied to minimize the vulnerability of the smart contract. The model-driven smart contract development method is an addition to the knowledge base as smart contract development needs a unique form of economic and defensive thinking, which is facilitated through the described method. The time-costliness may currently be a disadvantage, but the construction of the different artifacts may prove useful when the business knowledge can be re-used when porting smart contracts to other blockchain platforms.

8. Discussion

This research has been conducted with the aim of answering the main research question in the best manner possible. However, research is always prone to certain limitations and threats to the validity of research. In this chapter, we will reflect upon the threats to the validity and discuss the limitations in order to show how the quality of this research has been ensured. When evaluating research, a distinction between three main criteria can be made, namely internal validity, external validity, and reliability [107]. In the coming sections, these concepts and how they relate to this research are explored.

8.1 Internal Validity

Research can be called internally valid if the relations described between constructs are valid. In this research, an extensive literature research has been conducted in order to define the concepts and to determine if there is a relationship between the concepts to create the model-driven smart contract development method. To the best of our knowledge, all available knowledge has been applied to construct the method, but research on smart contracts is relatively new, and there is a high output of new research scattered in multiple venues. The method, however, is not prone to such rapid change, as all concepts used in the creation of the method find their foundation in earlier work than smart contracts development.

The method has been evaluated by both a case study and an experiment. By constructing the method with knowledge gained through a literature review and evaluating it through a case study and an experiment, the application of triangulation has been attempted. Triangulation refers to the use of multiple methods to develop a comprehensive understanding of phenomena [108]. By doing so, both the practical working of the method, as well as the underlying assumption on comprehension have been evaluated. However, the evaluation approaches do have their limitations.

The case study has an important limitation by not having actual smart contract developers involved in the execution. As smart contract development is a novel field, there is a lack of experts which are therefore missing from the case study. This does mean that the creator of the method performed the case study, and subsequently interpreted the findings of this case study. It would be better if the actual stakeholders

at whom the method is aimed would be involved in a case study, yielding well-founded insights from their perspective.

The experimental evaluation is also prone to validity threats. To counter the influence of events that occur between measurements, maturation, and the effects of testing, the choice for a static group comparison was made. This has as a consequence that selection bias and the capabilities of participants can have a large effect on the outcome. The relatively small number of participants only makes this threat larger. However, in chapter 6.4 it is described how participants were homogenous in terms of characteristics, the number of participants in each group was equal, and participants were randomly assigned over the two groups in order to mitigate these threats. The results of the experiment showed the most important threat to the validity of the experimental evaluation, namely the instrumentation. The results showed that the measurement instrument might not correctly reflect the participants' comprehension. This would mean the entire experiment can be deemed invalid. However, in assessing the validity of the instrumentation, results also showed that for the second scenario there was far littler discrepancy between the quantitative and qualitative data. Both scenarios showed similar results, meaning that it would be too blunt to disqualify all the conclusions drawn from the experimental evaluation.

8.2 External Validity

The external validity of research refers to the extent to which the results are generalizable to other contexts. The model-driven smart contract development method is aimed at a wide variety of smart contract development projects. The only condition for its implementation is that the smart contract will have the structure of a state machine. This means the method itself is externally valid in terms of applicability.

If functionality needs to be added to a smart contract, this can be done relatively easily by adding ADICO statements to the existing documentation, and subsequently adding the new states to the existing FSM. This means that the model is externally valid in terms of scalability.

The evaluation of the methods finds the most threats to external validity in this research. The case study serves as a demonstration but did not include validation

from external sources. This means it could be prone to researcher bias, as the same researcher who constructed the method, validated it in its entirety. To counter this, an experiment was conducted in which assumptions made in the construction of the method were validated. The biggest threat to the external validity in the experiment comes from the subject group. This group was homogeneous in order to make a valid comparison, but the heterogeneity lowers the generalizability. The participants were all higher-educated and have an affinity with IT systems and processes. However, the participants did not have any prior knowledge of smart contract development and the modeling methods presented to them, which somewhat mitigates this threat to the validity. The participants in the experiment showed that the modelling languages do not have a steep learning curve, making the method valid in terms of usability. However, not including actual stakeholders in the case study makes it that this threat is not completely mitigated.

8.3 Reliability

Reliability can be can be defined as the extent to which results are consistent over time and an accurate representation of the total population under study [109]. In order to guarantee the reliability of the study, a thorough documentation of the research approach has been made, which can be found in chapter 2. In the process of constructing the method, all choices have been made on the basis of existing frameworks like the MDA. The evaluation of the method has also been thoroughly documented in chapter 6, as to ensure the reliability.

To ensure that in the experimental evaluation the measurement instrument was reliable, an additional measurement of the comprehension has been done through post-experiment interviews. This additional evaluation showed that the comprehension metric was prone to invalidity caused by the assumptions of the participants. The reliability of the experimental evaluation therefore becomes somewhat dubious.

9. Future Work

This research present the model-driven smart contract development method as its main artifact. The research presented in this thesis can be extended upon in several directions in the future. First, the focus in this research was on both gathering requirements, constructing, and evaluating the model-driven smart contract development method. Although there has been an evaluation, the context of this research leaves a gap in terms of further evaluation of the components. Future research could fill this gap by doing a more extensive experiment or applying the method in practice with domain experts and smart contract developers. In a more extensive experiment with for instance a more heterogeneous pool of participants or a before-after treatment, the benefits of the method may become even more apparent. Lessons learned from further evaluation could also extend the method in ways the current evaluation did not allow for.

Second, the method in its current form is not specialized or tailored to a specific application domain. Future research could specialize the method, allowing for fully automated transformations between the methods' models. These domains may become more apparent as the usage of smart contracts becomes a more mainstream practice. Further research can also be done into specific components of the method. For instance, if the PSM is studied intensively, automated ways to evaluate a contracts' security vulnerabilities can be identified.

Third, this research is focused on Ethereum in combination with Solidity. At the moment of writing this is by far the most widely used combination of programming language and execution platform, but in the highly volatile environment of blockchain platforms, this might rapidly change. Future research could focus on the transformations from the PIM as described in this research to several PSMs. By applying the method to multiple PSMs, the true advantage of a model-driven approach will become apparent, as the re-usage of knowledge allows the developer to quickly adapt to whatever platform will gain momentum.

10. References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic payment system," *J. Gen. Philos. Sci.*, vol. 39, no. 1, pp. 53–67, 2008.

[2] Nomura Research Institute, "Survey on Blockchain Technologies and Related Services," *Res. Rep.*, no. March, pp. 1–78, 2016.

[3] M. Conoscenti, A. Vetro, and J. C. De Martin, "Blockchain for the Internet of Things: A systematic literature review," *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl. AICCSA*, pp. 1–6, 2017.

[4] A. Bahga and V. K. Madisetti, "Blockchain Platform for Industrial Internet of Things," *J. Softw. Eng. Appl.*, vol. 09, no. 10, pp. 533–546, 2016.

[5] D. Drescher, *Blockchain basics: A non-technical introduction in 25 steps*. 2017.

[6] "Cryptocurrency Market Capitalizations." [Online]. Available: http://www.coinmarketcap.com/.

[7] C. Caginalp and G. Caginalp, "Valuation, liquidity price, and stability of cryptocurrencies," *Proc. Natl. Acad. Sci.*, vol. 115, no. 6, pp. 1131–1134, 2018.

[8] A. Nelson, "Cryptocurrency Regulation in 2018: Where the World Stands Right Now," *Bitcoin Mag.*, 2018.

[9] N. Szabo, "Smart Contracts: Building Blocks for Digital Markets," 1997.

[10] F. Kinley, "A $50 MILLION HACK JUST SHOWED THAT THE DAO WAS ALL TOO HUMAN," *Wired.com*, 2016.

[11] M. Alharby and A. van Moorsel, "Blockchain Based Smart Contracts : A Systematic Mapping Study," *Comput. Sci. Inf. Technol. (CS IT)*, pp. 125–140, 2017.

[12] D. W. Cearley, M. J. Walker, B. Burke, and S. Searle, "Top 10 Strategic Technology Trends for 2017: A Gartner Trend Insight Report Insight From the Analyst Strategic Technology Trends — Threat or Opportunity?," no. March, 2017.

[13] T. Chen, X. Li, X. Luo, and X. Zhang, "Under-optimized smart contracts devour your money," *SANER 2017 - 24th IEEE Int. Conf. Softw. Anal. Evol. Reengineering*, pp. 442–446, 2017.

[14] "Etherscan." [Online]. Available: http://www.etherscan.io/.

[15] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9604 LNCS, pp. 79–94, 2016.

[16] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making Smart Contracts Smarter," *Proc. 2016 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS'16*, pp. 254–269, 2016.

[17] R. S. Pressman, *Software Engineering: A Practicioner's Approach*. 2001.

[18] W. Scacchi, "Process Models in Software Engineering," *Encycl. Softw. Eng.*, no. May, pp. 1–24, 2001.

[19] T. Stahl, M. Völter, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. 2006.

[20] M. Afonso, R. Vogel, and J. Teixeira, "From code centric to model centric software engineering: Practical case study of MDD infusion in a systems integration company," *Proc. - Jt. Meet. 4th Work. Model. Dev. Comput. Syst. 3rd Int. Work. Model. Methodol. Pervasive Embed. Software, MBD/MOMPES 2006*, pp. 125–134, 2006.

[21] S. T. March and G. F. Smith, "Design and natural science research on information technology," *Decis. Support Syst.*, vol. 15, pp. 251–266, 1995.

[22] H. A. Simon, *The sciences of the artificial, (third edition)*, vol. 33, no. 5. 1997.

[23] M. Silver, M. Markus, and C. Beath, "The information technology interaction model: A foundation for the MBA core course," *MIS Q.*, pp. 361–390, 1995.

[24] A. Hevner, S. March, and J. Park, "Design Science in Information Systems Research," *MIS Q. Manag. Inf. Syst.*, vol. 28, no. 1, pp. 75–105, 2004.

[25] A. R. Hevner, "A Three Cycle View of Design Science Research," *Scand. J. Inf. Syst.*, vol. 19, no. 2, pp. 87–92, 2007.

[26] R. Baskerville, "What design science is not," *Eur. J. Inf. Syst.*, vol. 17, no. 5, pp. 441–443, 2008.

[27] A. N. S. Contract and D. A. Platform, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," 2009.

[28] M. Swan, *Blockchain: Blueprint for a New Economy*. 2015.

[29] J. Leibowitz, "Bitcoin: A 21st Century Currency Explained By a Wall Street Veteran - CoinDesk," *Coindesk*, 2016. [Online]. Available: http://www.coindesk.com/bitcoin-explained-global-currency-wall-street-veteran/. [Accessed: 09-Nov-2017].

[30] L. Lamport, R. Shostak, and M. Pease, "The Byzantine Generals Problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, 1982.

[31] C. Dannen, *Introducing ethereum and solidity: Foundations of cryptocurrency and blockchain programming for beginners*. 2017.

[32] W. Diffie, W. Diffie, and M. E. Hellman, "New Directions in Cryptography," *IEEE Trans. Inf. Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[33] H. J. Highland, "Security in computing," *Comput. Secur.*, vol. 16, no. 3, p. 181, 1997.

[34] Benjamine, "Encryption." 2017.

[35] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[36] B. K. Yi, "Digital signatures," *Computer*, 2006. [Online]. Available: http://www.revasolutions.com/EXPERTISE/PROCESS-MANAGEMENT/DIGITAL-SIGNATURES/.

[37] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*, vol. 19964964. 1996.

[38] R. C. Merkle, "One way hash functions and DES," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 435 LNCS, pp. 428–446, 1990.

[39] L. P. Prieto, M. J. Rodríguez-Triana, M. Kusmin, and M. Laanpere, "Smart school multimodal dataset and challenges," *CEUR Workshop Proc.*, vol. 1828, pp. 53–59, 2017.

[40] G. Fox, "Peer-to-peer networks," *Comput. Sci. Eng.*, vol. 3, no. 3, pp. 75–77, 2001.

[41] H. Yan, "Distributed File Systems," *Most*, pp. 1–12, 2008.

[42] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," *Proc. - 2017 IEEE 6th Int. Congr. Big Data, BigData Congr. 2017*, no. October, pp. 557–564, 2017.

[43] M. Biella and V. Zinetti, "Blockchain Technology and Applications from a Financial Perspective," *Unicredit*, 2016.

[44] D. Liu and J. Camp, "Proof of Work can Work," p. 16, 2006.

[45] I.-C. Lin and T.-C. Liao, "A Survey of Blockchain Security Issues and Challenges," *Int. J. Netw. Secur.*, vol. 1919, no. 55, pp. 653–659, 2017.

[46] K. Christidis and M. Devetsikiotis, “Blockchains and Smart Contracts for the Internet of Things,” *IEEE Access*, vol. 4, pp. 2292–2303, 2016.

[47] Josh Stark, “Making Sense of Blockchain Smart Contracts,” *Coindesk.Com*, 2016. [Online]. Available: http://www.coindesk.com/making-sense-smart-contracts/. [Accessed: 24-Nov-2017].

[48] V. Morabito, “Business Innovation Through Blockchain,” pp. 101–124, 2017.

[49] W. P. Government of Canada, “White Paper,” *Aborig. Policy Stud.*, vol. 1, no. 1, pp. 1–36, 2011.

[50] M. Bartoletti and L. Pompianu, “An Empirical analysis of smart contracts: Platforms, applications, and design patterns,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10323 LNCS, pp. 494–509, 2017.

[51] M. Andrychowicz, S. Dziembowski, D. Malinowski, and Ł. Mazurek, “Secure multiparty computations on bitcoin,” *Proc. - IEEE Symp. Secur. Priv.*, vol. 59, no. 4, pp. 443–458, 2014.

[52] I. Bentov and R. Kumaresan, “How to use Bitcoin to design fair protocols,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8617 LNCS, no. PART 2, pp. 421–439, 2014.

[53] M. Alharby and A. van Moorsel, “Blockchain Based Smart Contracts : A Systematic Mapping Study,” *Comput. Sci. Inf. Technol. (CS IT)*, vol. 4, no. 4, pp. 125–140, 2017.

[54] B. Marino and A. Juels, “Setting standards for altering and undoing smart contracts,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9718, pp. 151–166, 2016.

[55] F. Idelberger, G. Governatori, R. Riveret, and G. Sartor, “Evaluation of logic-based smart contracts for blockchain systems,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9718, J. J. Alferes, L. Bertossi, G. Governatori, P. Fodor, and D. Roman, Eds. Cham: Springer International Publishing, 2016, pp. 167–183.

[56] A. Brown, “An introduction to Model Driven Architecture,” *Dev. Work. Libr.*, no. Part I, pp. 1–16, 2004.

[57] B. Hailpern and P. Tarr, “Model-driven development: The good, the bad, and the ugly,” *IBM Syst. J.*, vol. 45, no. 3, pp. 451–461, 2006.

[58] H. Stachowiak, *Allgemeine Modelltheorie*. 1973.

[59] J. Ludewig, “Models in software engineering – an introduction,” *Softw Syst Model Digit. Object Identifier*, vol. 2, pp. 5–14, 2003.

[60] J. Bèzivin, F. Jouault, and P. Valduriez, “On the need for megamodels,” *Work. Best Pract. Model. Softw. Dev. 19th Annu. ACM Conf. Object-Oriented Program. Syst. Lang. Appl.*, no. 1, pp. 1–9, 2004.

[61] T. Kühne, “Matters of (meta-) modeling,” *Softw. Syst. Model.*, vol. 5, no. 4, pp. 369–385, 2006.

[62] T. Bloomfield, “MDA, meta-modelling and model transformation: Introducing new technology into the defence industry,” in *European Conference on Model Driven Architecture-Foundations and Applications*, 2005, pp. 9–18.

[63] D. Shirtz, M. Kazakov, and Y. Shaham-Gafni, “Adopting model driven development in a large financial organization,” in *European Conference on Model Driven Architecture-Foundations and Applications*, 2007, pp. 172–183.

[64] J. C. A. Ferreira, “MDAI: Model based design in automobile industry,” in *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*, 2009, pp. 434–439.

[65] P. Baker, S. Loh, and F. Weil, “Model-Driven engineering in a large industrial context—motorola case study,” in *International Conference on Model Driven Engineering Languages and Systems*, 2005, pp. 476–491.

[66] A. MacDonald, D. Russell, and B. Atchison, “Model-driven development within a legacy system: an industry experience report,” in *Software Engineering Conference, 2005. Proceedings. 2005 Australian*, 2005, pp. 14–22.

[67] A. Mattsson, B. Lundell, B. Lings, and B. Fitzgerald, “Experiences from representing software architecture in a large industrial project using model driven development,” in *Proceedings of the Second Workshop on SHAring and Reusing architectural Knowledge Architecture, Rationale, and Design Intent*, 2007, p. 6.

[68] R. France and B. Rumpe, “Model-Driven Development of Complex Software: A Research Roadmap,” *Futur. Softw. Eng. 2007 ICSE.*, no. May 2007, pp. 37–54, 2007.

[69] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, vol. 1, no. 1. 2012.

[70] OMG, “OMG MDA Guide rev. 2.0,” *OMG Doc. ormsc*, vol. 2.0, no. June, pp. 1–15, 2014.

[71] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and J. G. Timothy, *Eclipse Modeling Framework: A Developer’s Guide*. 2003.

[72] JMI, “Java Metadata Interface (JMI),” *Sun Microsystems, Inc.*, 2002.

[73] M. P. Singh, “Ontology for commitments in multiagent systems: toward a unification of normative concepts,” *Artif. Intell. Law*, vol. 7, no. 1, pp. 97–113, 1999.

[74] S. Cook, *Domain-Specific modeling and model driven architecture*. 2004.

[75] R. Paige, “Model-driven software development. By Thomas Stahl and Markus Volter. Published by John Wiley & Sons, New York, 2006. ISBN: 0470025700, 444 pages. Price £33.99. Soft Cover,” *Softw. Testing, Verif. Reliab.*, vol. 18, no. 4, pp. 251–252, 2008.

[76] N. Debnath, M. C. Leonardi, M. V. Mauco, G. Montejano, and D. Riesco, “Improving model driven architecture with requirements models,” *Proc. - Int. Conf. Inf. Technol. New Gener. ITNG 2008*, pp. 21–26, 2008.

[77] Object Management Group, “Unified Modeling Language v2.5.1,” no. December, 2017.

[78] S. J. Meller, *MDA Distilled: Principles of Model-Driven Architecture*. Addison-Wesley, 2004.

[79] Object Management Group, “META OBJECT FACILITY SPECIFICATION VERSION 2.5.1,” 2016. .

[80] C. K. Frantz and M. Nowostawski, “From institutions to code: Towards automated generation of smart contracts,” *Proc. - IEEE 1st Int. Work. Found. Appl. Self-Systems, FAS-W 2016*, pp. 210–215, 2016.

[81] S. E. S. C. E. Ostrom, “A Grammar of Institutions Sue E . S . Crawford ; Elinor Ostrom,” *Polit. Sci.*, vol. 89, no. 3, pp. 582–600, 2007.

[82] D. Helbing and S. Balietti, “How to Do Agent-Based Simulations in the Future : From Modeling Social Mechanisms to Emergent Phenomena and Interactive Systems Design Why Develop and Use Agent-Based Models ?,” *Time*, no. 11-6–24, pp. 1–55, 2011.

[83] H. V. Parunak, R. Savit, and R. L. Riolo, “Agent-based modeling vs. equation-based modeling: A case study and users’ guide,” *Proc. Multi-agent Syst. Agent-based Simul.*, pp. 10–25, 1998.

[84] C. K. Frantz *et al.*, “PRIMA 2013: Principles and Practice of Multi-Agent Systems,” vol. 8291, no. December, 2013.

[85] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, “Untrusted Business Process Monitoring and Execution,” *Int. Conf. Bus. Process Manag.*, pp. 329–347, 2016.

[86] Object Management Group, “BPMN 2.0 specification,” 2011. [Online]. Available: https://www.omg.org/spec/BPMN/2.0/.

[87] I. Weber and G. Governatori, “Using Blockchain to Enable Untrusted Business Process Monitoring and Execution,” no. June 2016, pp. 1–16.

[88] A. Mavridou and A. Laszka, “Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach,” 2017.

[89] Ethereum, “Solidity specification.” [Online]. Available: http://solidity.readthedocs.io/en/v0.4.24/.

[90] J. Krogstie, “Quality of Conceptual Models in Model Driven Software Engineering,” *Concept. Model. Perspect.*, pp. 185–198, 2017.

[91] S. Brinkkemper, “Method engineering: Engineering of information systems development methods and tools,” *Inf. Softw. Technol.*, vol. 38, no. 4 SPEC. ISS., pp. 275–280, 1996.

[92] F. Harmsen, S. Brinkkemper, and H. Oei, *Situational Method Engineering for Information System Project Approaches*, vol. 55, no. September. 1994.

[93] J. Ralyte, C. Rolland, J. Ralyté, and C. Rolland, “An assembly process model for method engineering To cite this version : HAL Id : hal-00707078 An assembly process model for method engineering,” 2012.

[94] J. Ralyté, R. Deneckère, and C. Rolland, “Towards a Generic Model for Situational Method Engineering Generic Process Model for Situational Method Engineering,” *Adv. Inf. Syst. Eng.*, pp. 95–110, 2003.

[95] M. Dumas and A. H. M. Hofstede, “UML Activity Diagrams as a Workflow Specification Language,” *Proc. 4th Int. Conf. Unified Model. Lang. Model. Lang. Concepts, Tools*, pp. 76–90, 2001.

[96] H. C. Purchase, L. Colpoys, D. Carrington, and M. McGill, “UML class diagrams: an empirical study of comprehension,” vol. 9, no. December, pp. 149–178, 2003.

[97] I. van de Weerd and S. Brinkkemper, “Meta-Modeling for Situational Analysis and Design Methods,” *Handb. Res. Mod. Syst. Anal. Des. Technol. Appl.*, pp. 35–54, 2008.

[98] B. Nuseibeh and S. Easterbrook, “Requirements Engineering: A Roadmap, IEEE,” *Foundations*, vol. 1, pp. 35–46.

[99] D. Hutchison and J. C. Mitchell, *Dependability Metrics Advanced Lectures*. 1973.

[100] V. R. Basili, “Applying the Goal/Question/Metric paradigm in the experience factory,” *Softw. Qual. Assur. Meas. A Worldw. Perspect.*, vol. 7, no. 4, pp. 21–44, 1993.

[101] V. R. Basili, G. Caldiera, and D. Rombach, "The goal question metric approach," *Encycl. Softw. Eng.*, vol. 1, 1994.

[102] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*, vol. 9783642290. 2012.

[103] J. C. Campbell, D. T.; Stanley, "Experimental and Quasi-Experimental Designs for Research," *Handb. Res. teaching.*, pp. 1–71, 1963.

[104] N. A. Giannatasio, "Threats to validity of research designs," *Handb. Res. methods public Adm.*, pp. 145–165, 1998.

[105] E. Babbie, *The Practice of Social Research*. Wadsworth Cengage Learning, 2013.

[106] K. M. Ting, "Precision and Recall BT - Encyclopedia of Machine Learning," C. Sammut and G. I. Webb, Eds. Boston, MA: Springer US, 2010, p. 781.

[107] N. Golofshani, "Understanding reliability and validity in qualitative research," *Qual. Rep.*, vol. 8, no. 4, pp. 597–607, 2003.

[108] M. Q. Patton, "Enhancing the quality and credibility of qualitative analysis.," *Health Serv. Res.*, vol. 34, no. 5 Pt 2, pp. 1189–208, 1999.

[109] M. Bashir, M. T. Afzal, and M. Azeem, "Reliability and Validity of Qualitative and Operational Research Paradigm," *Pakistan J. Stat. Oper. Res.*, vol. 4, no. 1, p. 35, 2008.

Appendices

A. PDD of the Model-Driven Smart Contract Development Method

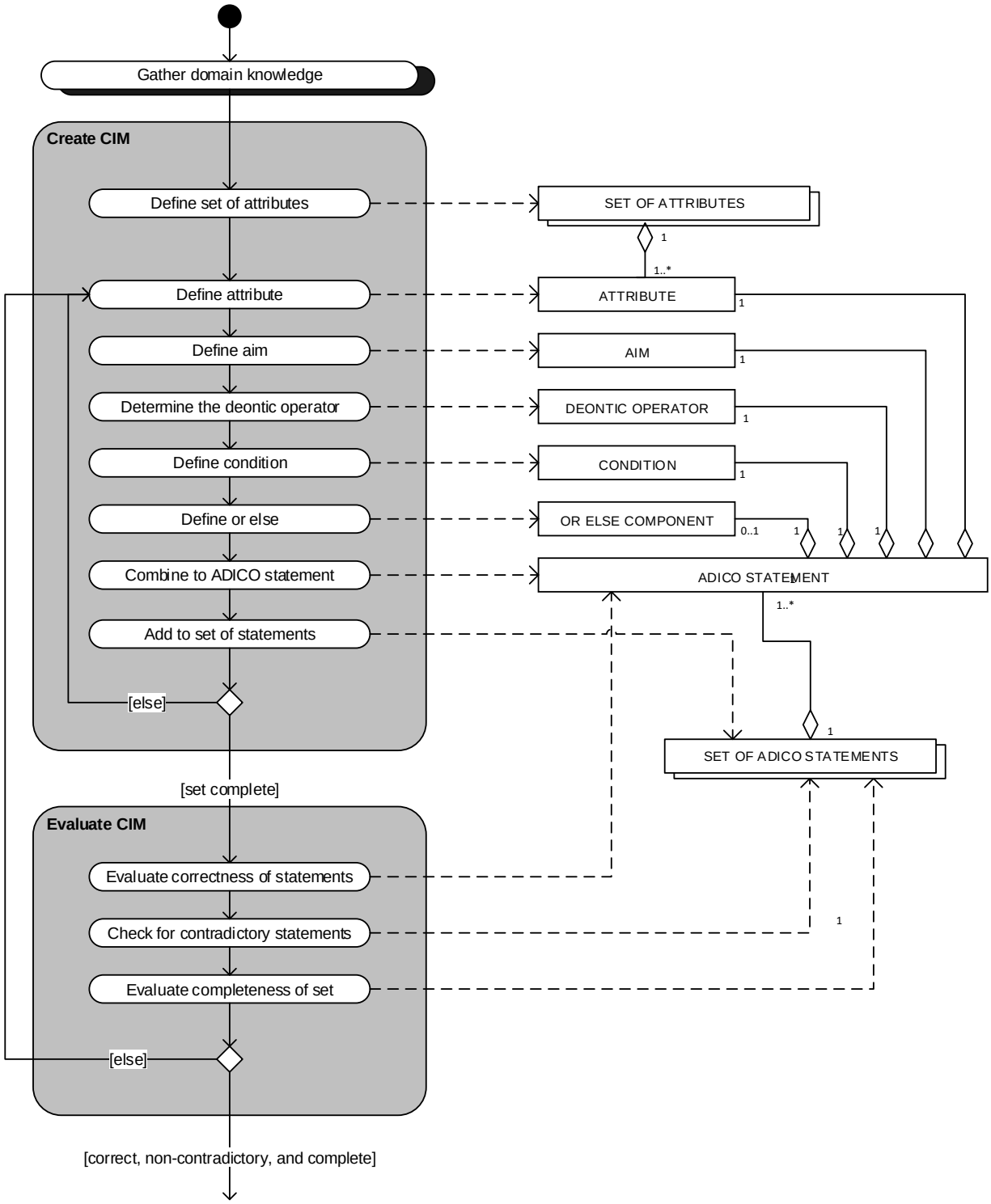


FIGURE 29: PDD TOTAL METHOD PART 1

Continued on the next page.

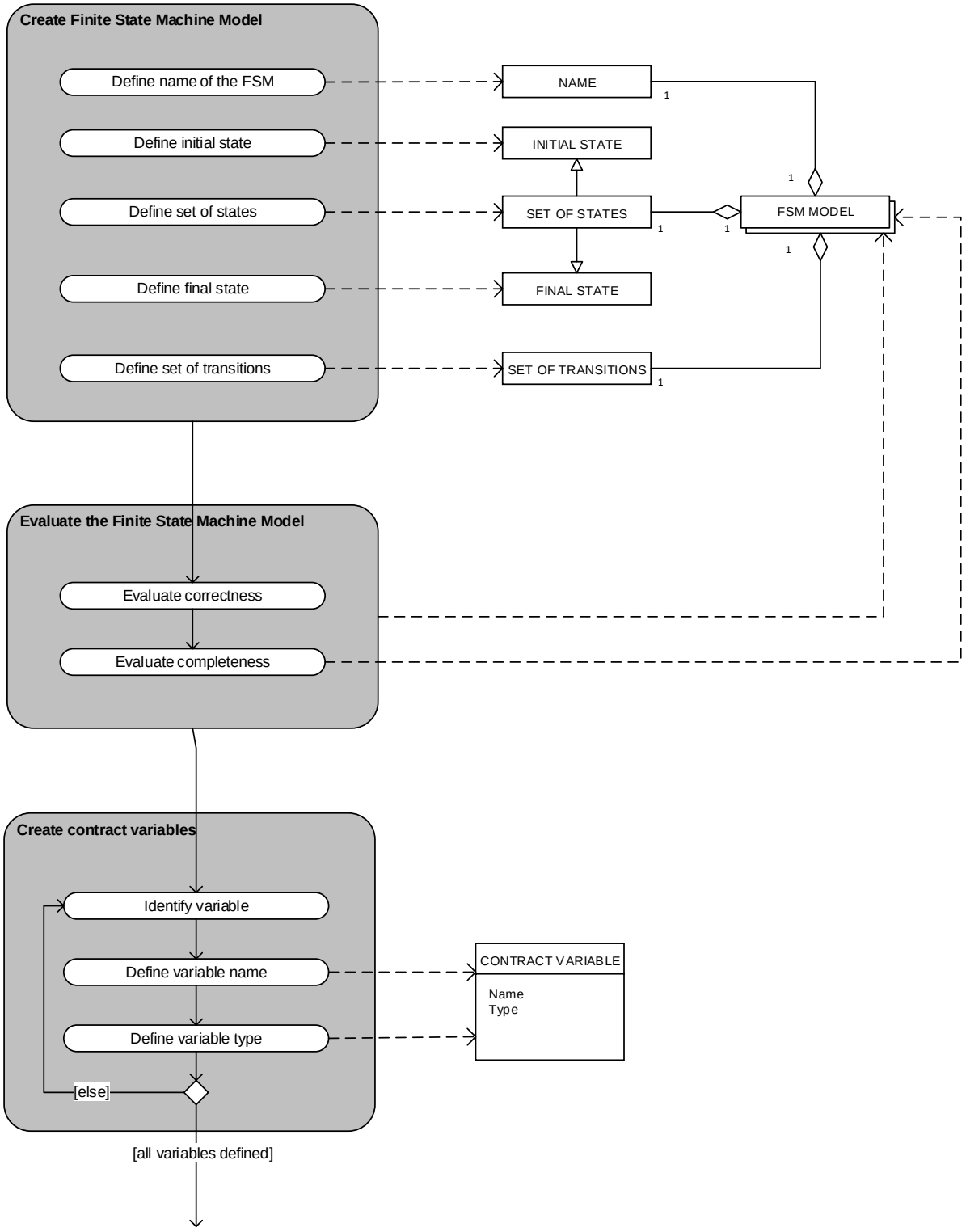


FIGURE 30: PDD TOTAL METHOD PART 2

Continued on the next page.

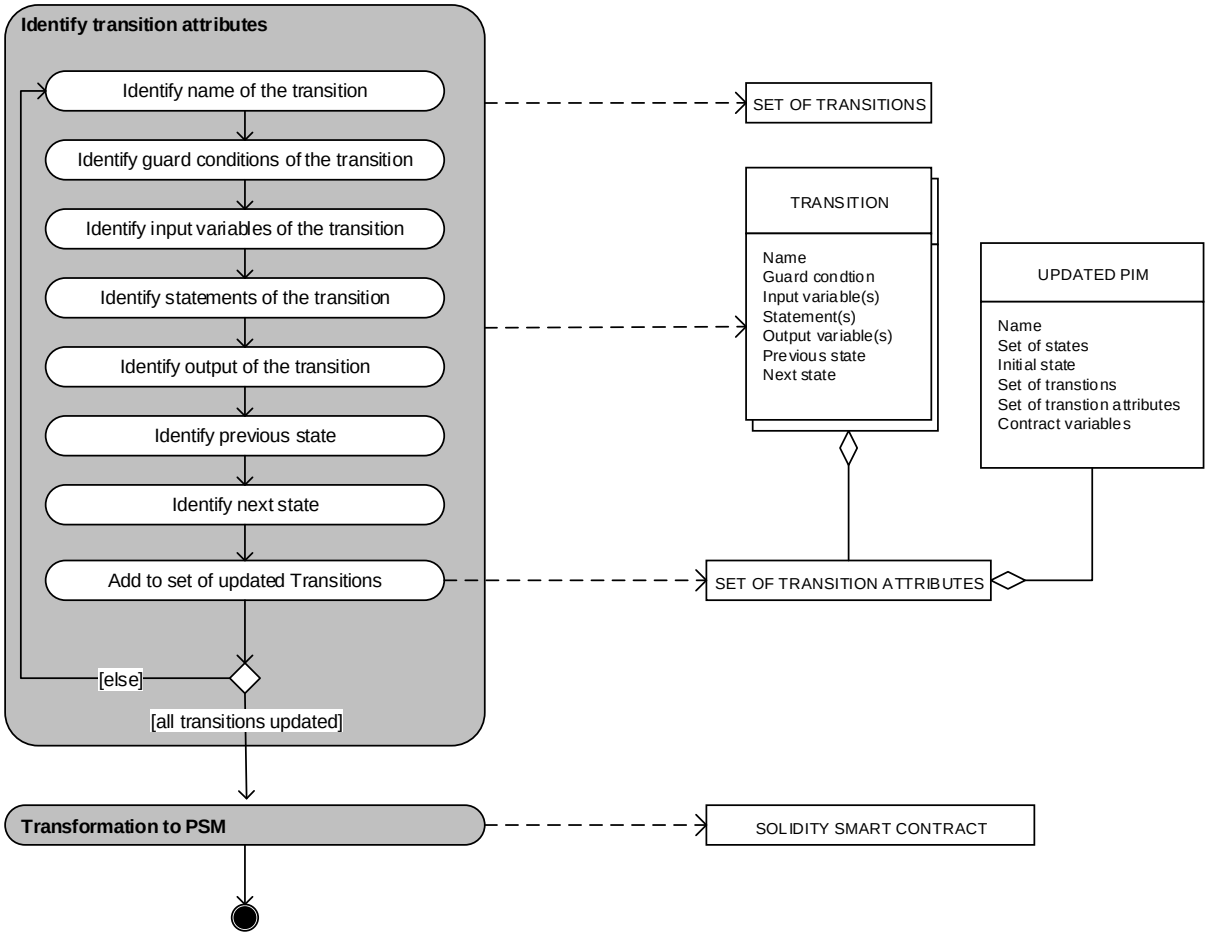
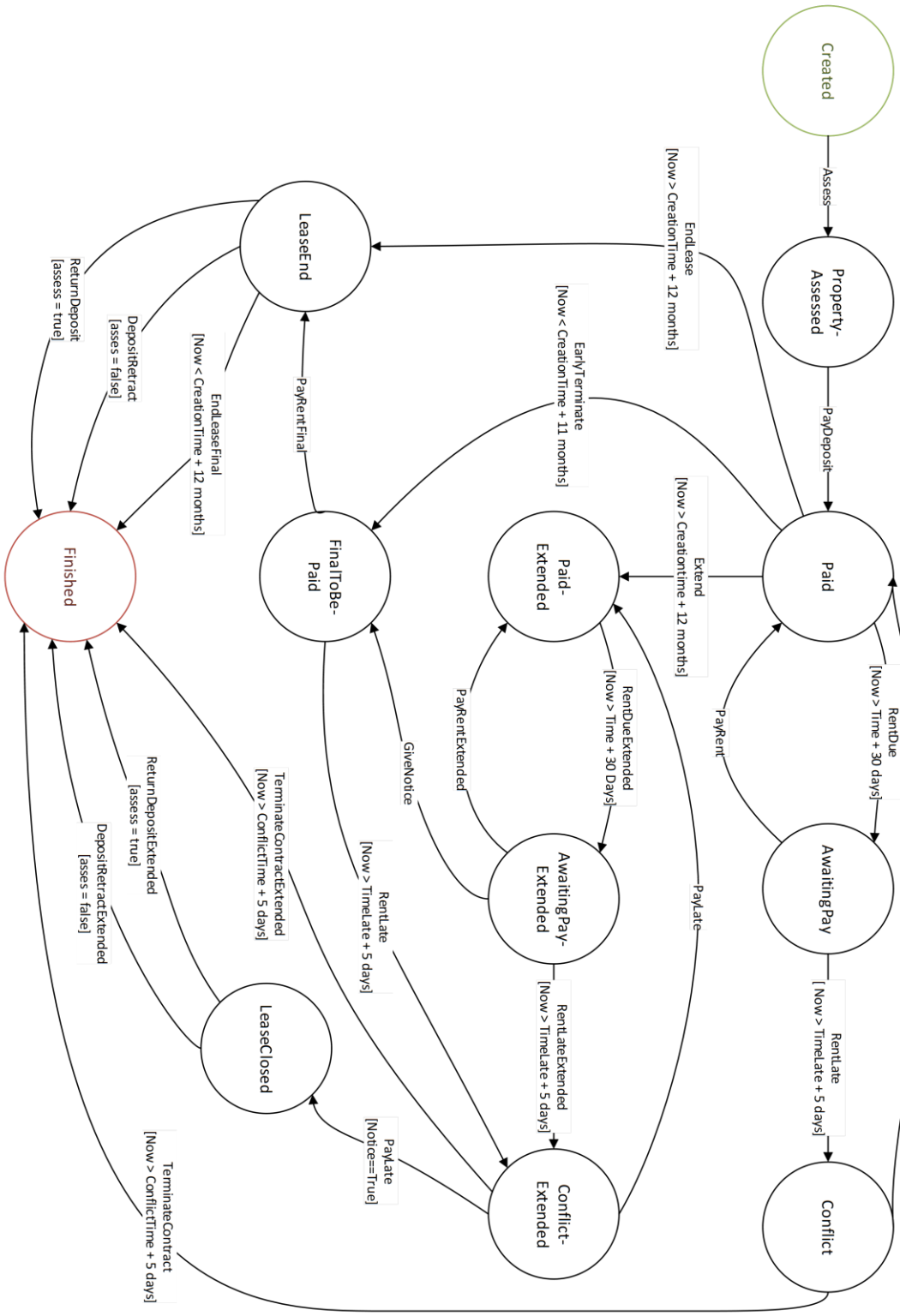


FIGURE 31: PDD TOTAL METHOD PART 3

B. Finite State Machine Model Lease Contract



C. Property Lease Smart Contract

```
pragma solidity ^0.4.0;

contract PropertyLease {

      //StatesDefinition
      enum States {Created, PropertyAssessed, Paid, AwaitingPay, Conflict,
PaidExtended, AwaitingPayExtended, ConflictExtended, FinalToBePaid,
LeaseEnd, LeaseClosed, Finished}
      States public state = States.Created;

      //VariablesDefinition
      uint public securityDeposit;
      uint public rent;
      uint public fine;
      uint public deposit;
      uint monthCounter;
      uint rentCounter;
      uint creationTime;

      bool private payDepositTrue;
      bool private notice;
      bool assess;

      address public landlord;
      address public tenant;
      address public propertymanager;

    // Locking
    bool private locked = false ;

    modifier locking {
        require (! locked );
        locked = true ;
        _;
        locked = false ;
    }

    // Transition counter
    uint private transitionCounter = 0;

    modifier transitionCounting ( uint nextTransitionNumber ) {
        require ( nextTransitionNumber == transitionCounter );
        transitionCounter += 1;
        _;
    }
```

```
      // Timed transtion
      modifier timedTransitions {
          if ((state == States.Paid) && (now >= rentCounter + 30 days))
          {state = States.AwaitingPay;}

          if ((state == States.AwaitingPay) && (now >= rentCounter + 35
days))
          {state = States.Conflict;}

          if ((state == States.PaidExtended) && (now >= rentCounter + 30
days))
          {state = States.AwaitingPayExtended;}

          if ((state == States.AwaitingPayExtended) && (now >= rentCounter
+ 35 days))
          {state = States.ConflictExtended;}

          _;
      }

      //Constructor function
    function PropertyLease(uint inputRent, uint inputFine) public {
          landlord = msg.sender;
          rent = inputRent;
          fine = inputFine;
          creationTime = now;
          securityDeposit = 2 * inputRent;
      }

    function Assess (uint nextTransitionNumber)
        locking
        transitionCounting ( nextTransitionNumber )
    {   require (state == States.Created);
        // Statements
        state == States.PropertyAssessed;
    }

    function PayDeposit (uint nextTransitionNumber)
        locking
        transitionCounting ( nextTransitionNumber )
    {   require (state == States.PropertyAssessed);
        // Statements
        state == States.Paid;
    }

    function RentDue (uint nextTransitionNumber)
        locking
        transitionCounting ( nextTransitionNumber )
        timedTransitions
    {   require (state == States.Paid);
        // Statements
        state == States.AwaitingPay;
    }

    function RentDueExtended (uint nextTransitionNumber)
        locking
        transitionCounting ( nextTransitionNumber )
        timedTransitions
    {   require (state == States.PaidExtended);
        // Statements
```

```
    state == States.AwaitingPayExtended;    }

function PayRent (uint nextTransitionNumber) payable
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.AwaitingPay);
    // Statements
    state == States.Paid;
}



function PayRentExtended (uint nextTransitionNumber) payable
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.AwaitingPayExtended);
    // Statements
    state == States.PaidExtended;
}

function PayRentFinal (uint nextTransitionNumber) payable
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.FinalToBePaid);
    // Statements
    state == States.LeaseEnd;
}

function RentLate (uint nextTransitionNumber)
    locking
    transitionCounting ( nextTransitionNumber )
    timedTransitions
{   require (state == States.AwaitingPay);
    // Statements
    state == States.Conflict;
}

function RentLateExtended (uint nextTransitionNumber)
    locking
    transitionCounting ( nextTransitionNumber )
    timedTransitions
{   require (state == States.AwaitingPayExtended);
    // Statements
    state == States.ConflictExtended;
}

function PayLate (uint nextTransitionNumber) payable
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.Conflict);
    // Statements
    state == States.Paid;
}

function PayLateExtended (uint nextTransitionNumber) payable
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.Conflict);
    // Statements
    state == States.Paid;
}
```

```
function PayLateFinal (uint nextTransitionNumber) payable
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.ConflictExtended);
    require (notice == true);
    // Statements
    state == States.Paid;
}



  function TerminateContract (uint nextTransitionNumber)
    locking
    transitionCounting ( nextTransitionNumber )
    timedTransitions
{   require (state == States.Conflict);
    // Statements
    state == States.Finished;
}

  function TerminateContractExtended (uint nextTransitionNumber)
    locking
    transitionCounting ( nextTransitionNumber )
    timedTransitions
{   require (state == States.ConflictExtended);
    // Statements
    state == States.Finished;
}

function Extend (uint nextTransitionNumber)
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.Paid);
    // Statements
    state == States.PaidExtended;
}

function EarlyTerminate (uint nextTransitionNumber)
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.Paid);
    // Statements
    state == States.FinalToBePaid;
}

function EndLease (uint nextTransitionNumber)
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.Paid);
    // Statements
    state == States.LeaseEnd;
}

function GiveNotice (uint nextTransitionNumber)
    locking
    transitionCounting ( nextTransitionNumber )
{   require (state == States.AwaitingPayExtended);
    // Statements
    state == States.FinalToBePaid;
}
```

```
    function ReturnDeposit (uint nextTransitionNumber)
        locking
        transitionCounting ( nextTransitionNumber )
    {   require (state == States.LeaseEnd);
        require (assess = true);
        // Statements
        state == States.Finished;
    }



    function DepositRetract (uint nextTransitionNumber)
        locking
        transitionCounting ( nextTransitionNumber )
    {   require (state == States.LeaseEnd);
        require (assess = false);
        // Statements
        state == States.Finished;
    }

    function ReturnDepositLate (uint nextTransitionNumber)
        locking
        transitionCounting ( nextTransitionNumber )
    {   require (state == States.LeaseClosed);
        require (assess = true);
        // Statements
        state == States.Finished;
    }

    function DepositRetractLate (uint nextTransitionNumber)
        locking
        transitionCounting ( nextTransitionNumber )
    {   require (state == States.LeaseClosed);
        require (assess = false);
        // Statements
        state == States.Finished;
    }

    function EndLeaseFinal (uint nextTransitionNumber)
        locking
        transitionCounting ( nextTransitionNumber )
    {   require (state == States.LeaseEnd);
        // Statements
        state == States.Finished;
    }

}
```

D. Experimental Materials

i. Information Sheets without the CIM

Cover sheet A

Thank you for participating in this experiment. Before we start, please make sure you have the following materials in front of you:

- Sheet 1: Cover sheet
- Sheet 2: Introduction to Finite State Machines
- Sheet 3: Description of the lease contract
- Sheet 4: Finite State Machine of the lease contract
- Sheet 5: Fill in the blanks
- A sheet or laptop to write down your answers
- Scenario 1
- Scenario 2

Please read the introduction to Finite State Machines modeling first (Sheet 2). When you are done, read the description of the case (Sheet 3. Then have a look at the lease contract model (Sheet 4).

Task 1: The task is described on sheet 5, try to fill in the blanks using sheet 1 – 4 as information sources.

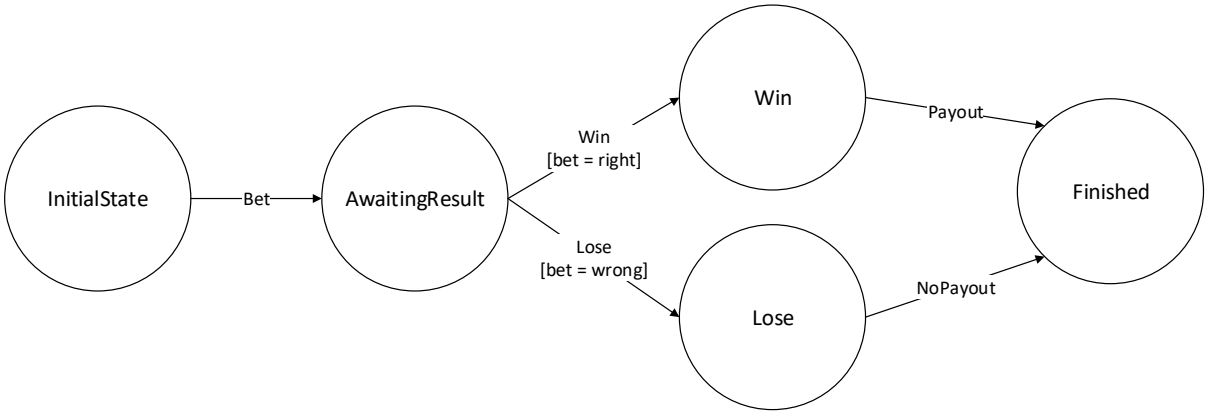
TASK 2: Have a look at the model named Scenario 1. In this version of the lease contract model, a scenario is highlighted by a red line. If a transition is used more than once, the red number shows how many times it was used. Use the finite state machine model description and the description of the case. Try to explain in your own words what the steps of the scenario are. Aim to be as complete as you can be, meaning that you try to describe every state and transition in as much detail as you can. When you are done with scenario 1, do the same for scenario 2.

Utrecht University

Introduction to finite state machines

In this experiment, you are going to analyse a finite state machine (FSM) model. This type of model is used to represent the behaviour of a system. FSMs show that, at any moment, a system is in a particular state. In such state, the system responds to certain stimuli or input, which lead to a change of the state. This change of state is called a transition.

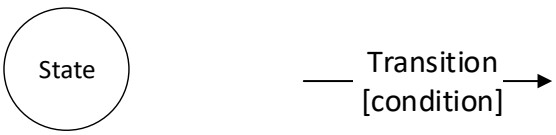
Depending on what stimulus or input the system gets, different transitions are followed (triggered), leading to different states. An FSM begins in an initial state, and the state evolves following the transitions that start from that state, depending on the stimulus/input. To make this concept clearer, FSMs are represented graphically, as shown in the following Figure.



A simple betting system is modelled. A circle stands for a state of the system. The arrows between the circles stand for transitions between the states. All the states and transitions have names, written in text.

The system starts in **the initial state**. A user makes a bet, transitioning the system to the state 'AwaitingResult'. The system will stay in this **state** until the result of the bet is in. After this, there are two options, either the user was right and wins the bet, or the user was wrong and loses the bet. Being wrong or right is a **condition**, which are written between [brackets] under the **transition** name. Conditions need to hold in order to enable a transition. Depending on what condition holds the system changes to a certain condition. In this case being right leads to the state 'Win' and consequently to the transition *Payout*, after which the system reaches the state 'Finished', **the final state**. Not all transitions have conditions; for instance, the *bet* transition is activated by the users' input.

So a model has at least an initial state, a final state, and a transition between these states. Circles are states, arrows are transitions between states, and conditions for transitions are shown between brackets under the transition name.



Utrecht University

Description of the contract

Have a look at the lease contract model. This represents an agreement between a landlord and a tenant through a smart contract. Instead of using a paper contract, the agreement is made electronically and by doing so, becomes a smart contract. Therefore, it can be modelled as a finite state machine. The situation is as follows:

First, the landlord launches the contract, after which the tenant pays a security deposit which corresponds to two months of rent. This security deposit is a safeguard against (i) possible damages to the property and (ii) the tenant not adhering to the terms of the agreement. The landlord and the tenant agree on an initial rental period of one year, with an option to extend the contract afterwards. If the tenant wants to terminate the agreement early, he/she loses his/her security deposit. After the initial period is over, if the lease renewal option is used, the agreement can be terminated each month. In this extended renting period, the tenant shall give a one-month notice if he wants to end the lease.

The rent is due every 30 days. The tenant has five days to pay the rent. After five days, the tenant gets a fine for his negligence. He then has another five days to pay the rent and the fine. If he does not pay this rent and fine after ten days of the original rent due date, the landlord has the right to terminate the agreement, and the tenant will not receive his security deposit back. In every other scenario, the rent can be late, but it has to be paid within a period of time with a fine, or else the landlord has the same rights.

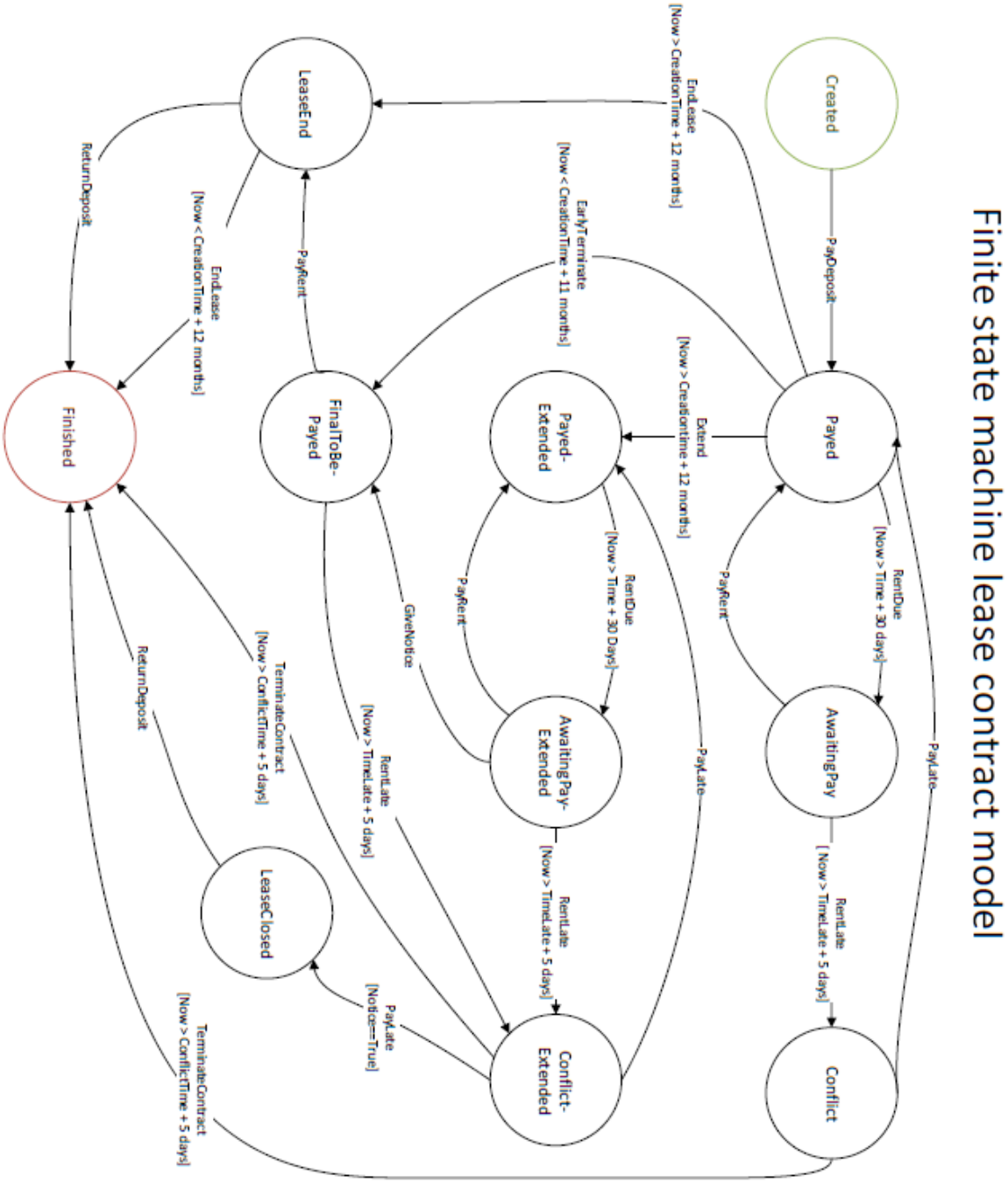
Utrecht University

Fill in the blanks

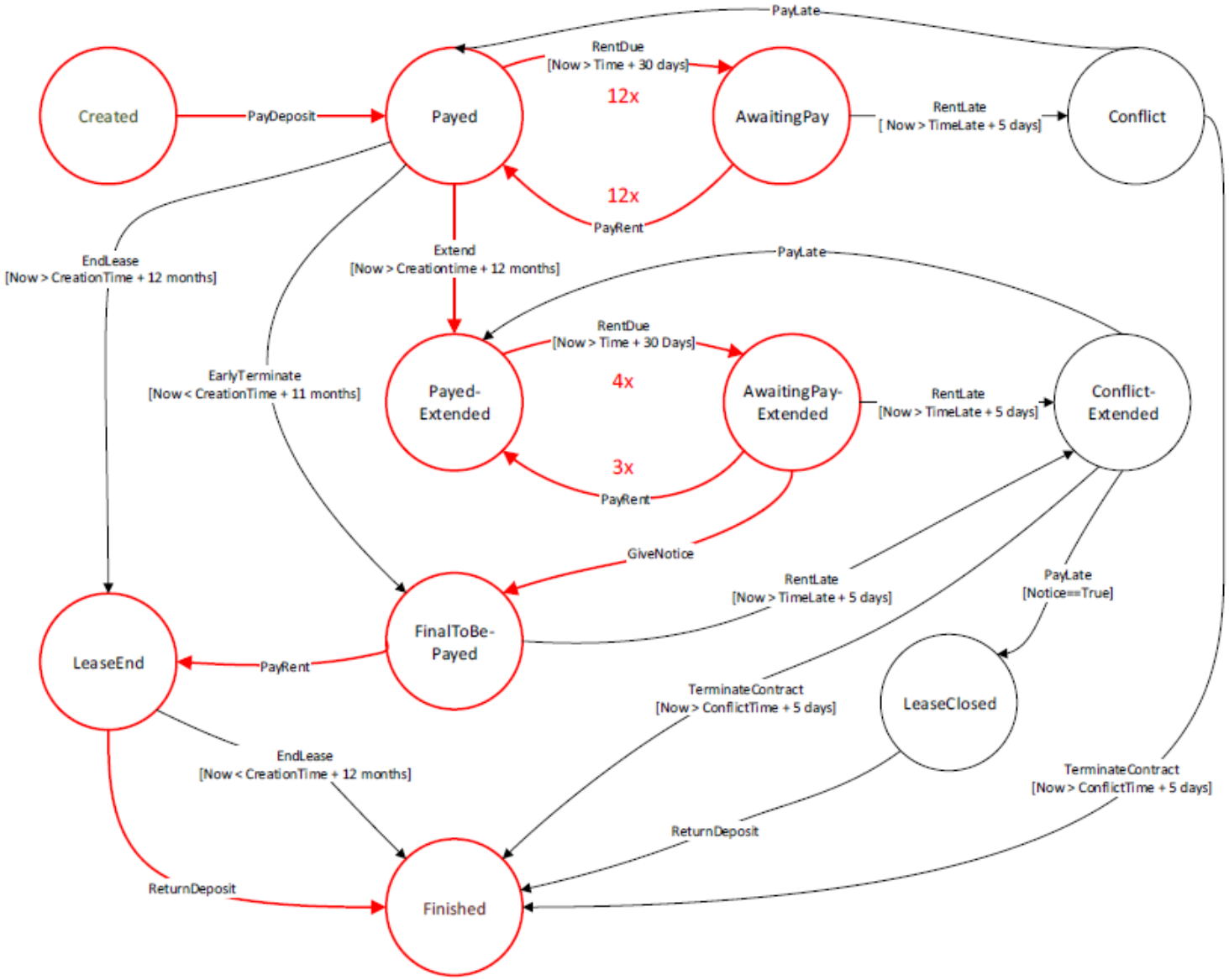
In this test, the materials you have in front of you are described through a number of statements. Some words in the statements have been blanked out. Using the materials, try to fill in the blanks.

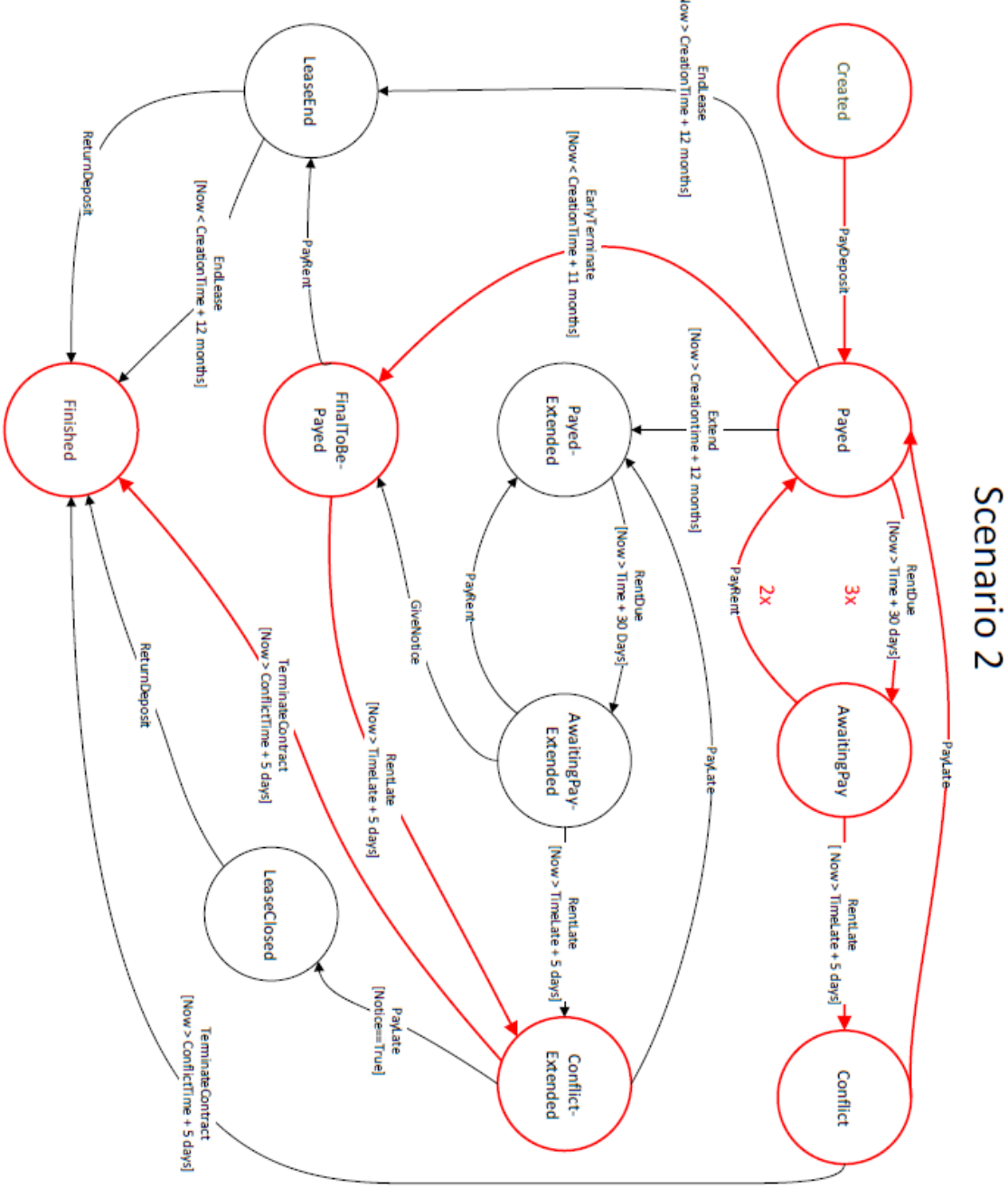
A finite state machine models the behavior of a system by showing transitions between (1) _____. Some transitions have a (2) _____ which needs to be fulfilled in order for the transition to happen, this is shown between brackets in the FSM. The FSM lease contract is a contract between two parties, namely the Landlord and the (3) _____. The first transition that can happen is the transition from the initial state 'Created' to the state (4) _____. This happens through the transition (5) _____, which is activated by the Tenant, who pays the (6) _____. The final state of the FSM lease contract is the state (7) _____.

ii. Models without the CIM



Scenario 1


Created
PayDeposit
Payed
RentDue
[Now > Time + 30 days]
12x
AwaitingPay
RentLate
[Now > TimeLate + 5 days]
Conflict
PayLate
12x
PayRent
EndLease
[Now > CreationTime + 12 months]
Extend
[Now > Creationtime + 12 months]
PayLate
RentDue
[Now > Time + 30 Days]
EarlyTerminate
[Now < CreationTime + 11 months]
Payed-
Extended
4x
AwaitingPay-
Extended
RentLate
[Now > TimeLate + 5 days]
Conflict-
Extended
3x
PayRent
GiveNotice
RentLate
[Now > TimeLate + 5 days]
PayLate
[Notice==True]
FinalToBe-
Payed
LeaseEnd
PayRent
TerminateContract
[Now > ConflictTime + 5 days]
LeaseClosed
EndLease
[Now < CreationTime + 12 months]
TerminateContract
[Now > ConflictTime + 5 days]
ReturnDeposit
ReturnDeposit
Finished


Scenario 2
Created
Payed
AwaitingPay
Conflict
LeaseEnd
Finished
FinalToBe-Payed
Payed-Extended
AwaitingPay-Extended
Conflict-Extended
LeaseClosed
PayDeposit
PayLate
RentDue
[Now > Time + 30 days]
RentLate
[Now > TimeLate + 5 days]
PayRent
2x
3x
EndLease
[Now > CreationTime + 12 months]
Extend
[Now > CreationTime + 12 months]
EarlyTerminate
[Now < CreationTime + 11 months]
ReturnDeposit
GiveNotice
[Now > TimeLate + 5 days]
TerminateContract
[Now > ConflictTime + 5 days]
PayLate
[Notice==True]

iii. Information Sheets with the CIM

Utrecht University

Cover sheet B

Thank you for participating in this experiment. Before we start, please make sure you have the following materials in front of you:

- Sheet 1: Cover sheet
- Sheet 2: Introduction to Finite State Machines
- Sheet 3: Description of the lease contract
- Sheet 4: Finite State Machine of the lease contract
- Sheet 5: ADICO statements of the lease contract
- Sheet 6: Fill in the blanks
- A sheet or laptop to write down your answers
- Scenario 1
- Scenario 2

Please read the introduction to Finite State Machines modeling first (Sheet 2). When you are done, read the description of the case (Sheet 3. Then have a look at the lease contract model (Sheet 4) and the ADICO statements (sheet 5).

Task 1: The task is described on sheet 6, try to fill in the blanks using sheet 1 – 5 as information sources.

TASK 2: Have a look at the model named Scenario 1. In this version of the lease contract model, a scenario is highlighted by a red line. If a transition is used more than once, the red number shows how many times it was used. Use the finite state machine model description, the description of the case and the ADICO statements. Try to explain in your own words what the steps of the scenario are. Aim to be as complete as you can be, meaning that you try to describe every state and transition in as much detail as you can. When you are done with scenario 1, do the same for scenario 2.

Utrecht University

Fill in the blanks

In this test, the materials you have in front of you are described through a number of statements. Some words in the statements have been blanked out. Using the materials, try to fill in the blanks.

A finite state machine models the behavior of a system by showing transitions between (1) _____.

Some transitions have a (2) _____ which needs to be fulfilled in order for the transition to

happen, this is shown between brackets in the FSM. The FSM lease contract is a contract between

two parties, namely the Landlord and the (3) _____. The first transition that can happen is the

transition from the initial state 'Created' to the state (4) _____. This happens through the

transition (5) _____, which is activated by the Tenant, who pays the (6) _____. The final

state of the FSM lease contract is the state (7) _____.

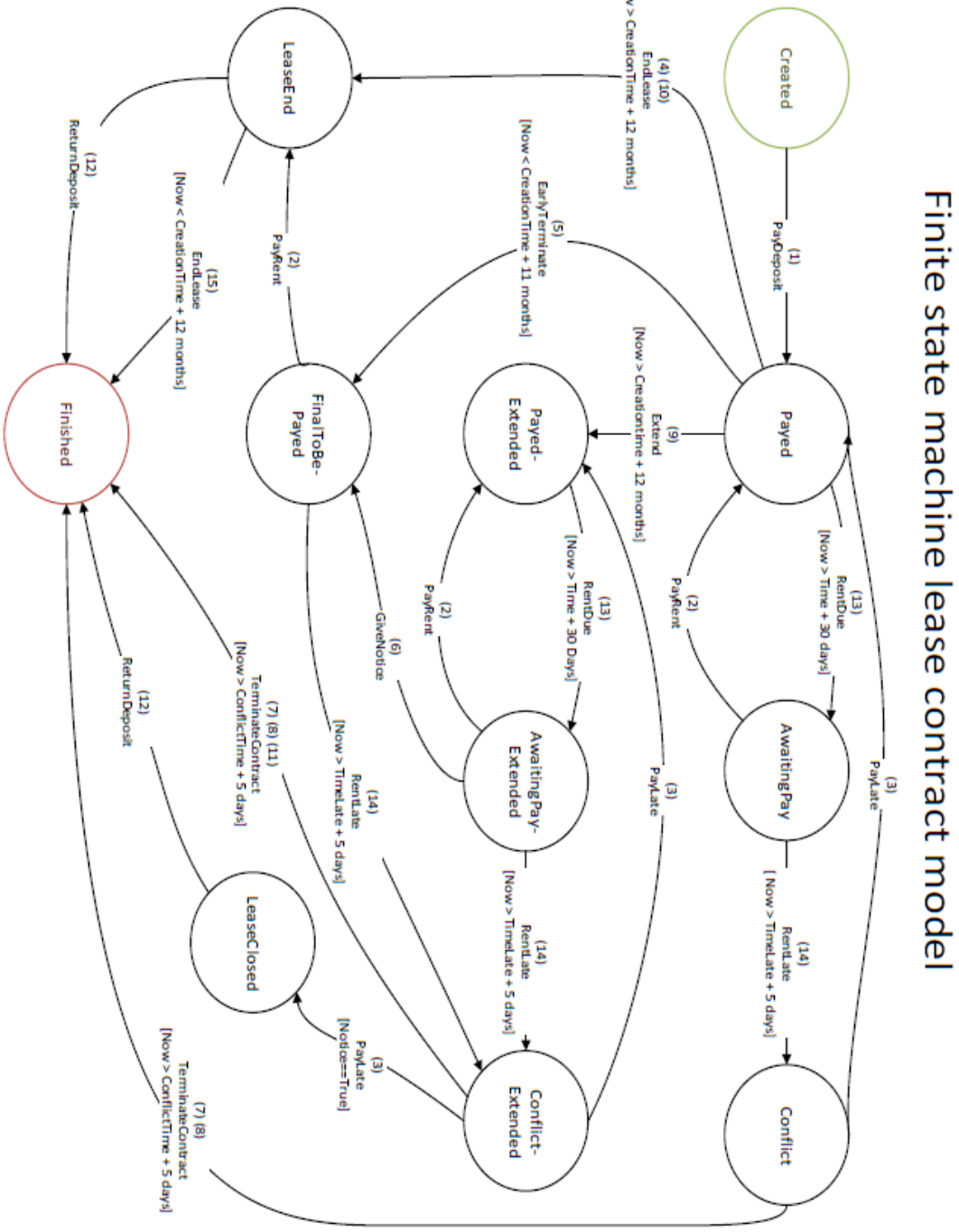
The components of an ADICO statement are an attribute, a deontic, an aim, a (8) _____, and an

or else. The ADICO statements and the (9) _____ are connected, because the transitions are

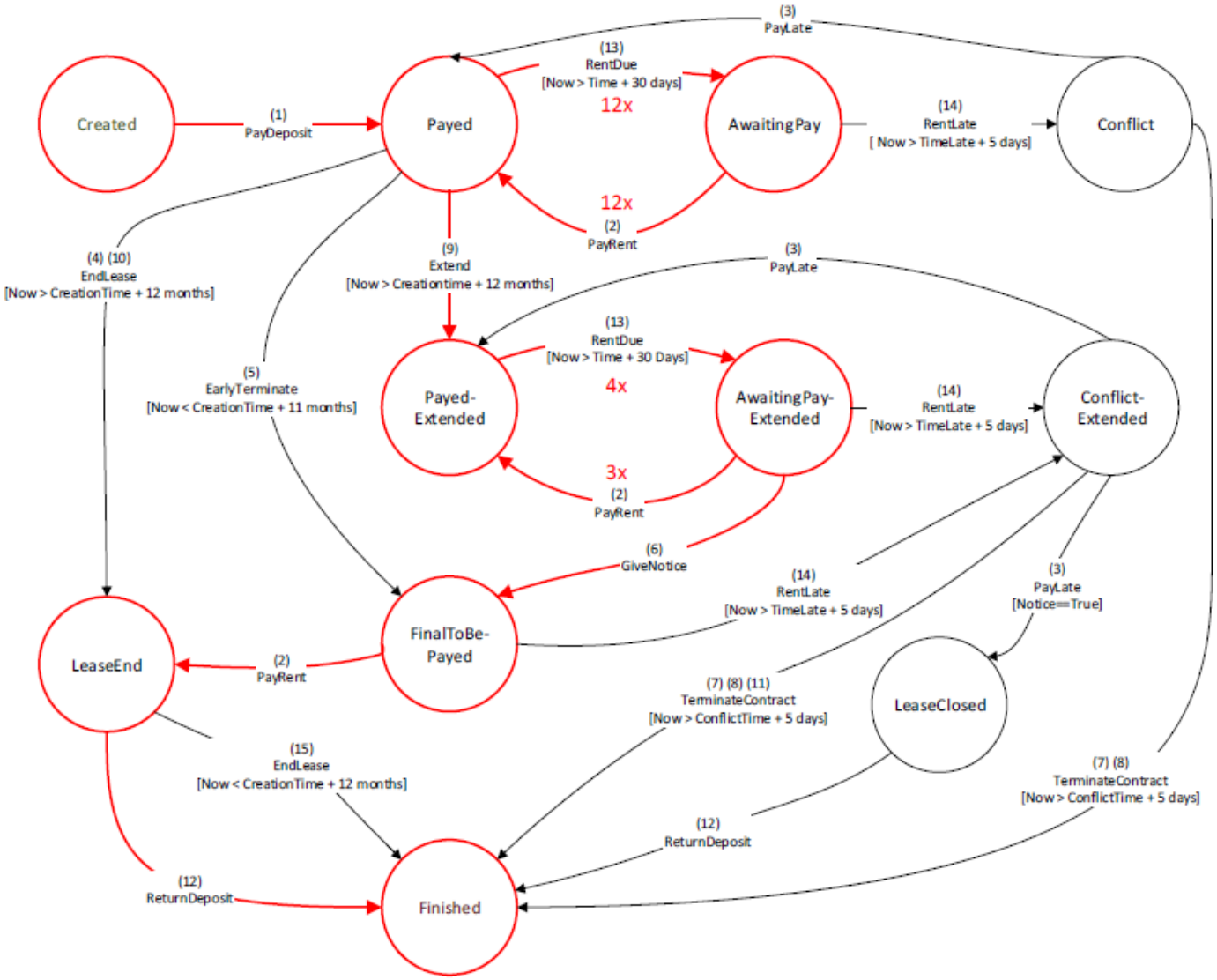
numbered (shown between parentheses), which relate to the numbers shown in the ADICO

statement table.

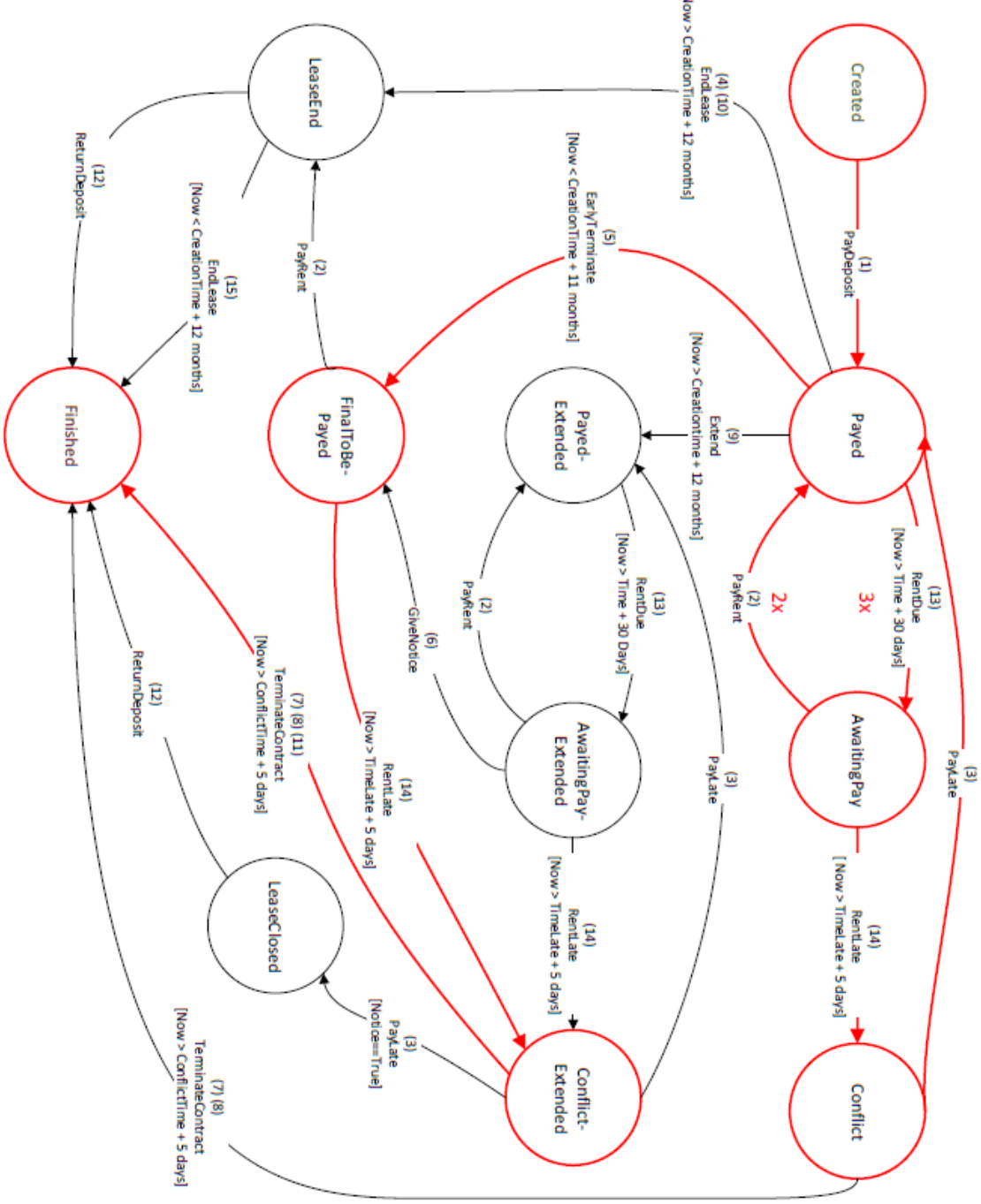
iv. Models with the CIM



Scenario 1


Created
(1)
PayDeposit
Payed
(13)
RentDue
[Now > Time + 30 days]
12x
AwaitingPay
(14)
RentLate
[Now > TimeLate + 5 days]
Conflict
(3)
PayLate
12x
(2)
PayRent
(4) (10)
EndLease
[Now > CreationTime + 12 months]
(9)
Extend
[Now > Creationtime + 12 months]
(3)
PayLate
(13)
RentDue
[Now > Time + 30 Days]
4x
Payed-
Extended
AwaitingPay-
Extended
(14)
RentLate
[Now > TimeLate + 5 days]
Conflict-
Extended
(5)
EarlyTerminate
[Now < CreationTime + 11 months]
3x
(2)
PayRent
(6)
GiveNotice
(14)
RentLate
[Now > TimeLate + 5 days]
(3)
PayLate
[Notice==True]
FinalToBe-
Payed
LeaseEnd
(2)
PayRent
LeaseClosed
(7) (8) (11)
TerminateContract
[Now > ConflictTime + 5 days]
(15)
EndLease
[Now < CreationTime + 12 months]
(7) (8)
TerminateContract
[Now > ConflictTime + 5 days]
(12)
ReturnDeposit
(12)
ReturnDeposit
Finished

Scenario 2


Created
(1)
PayDeposit
Payed
(3)
PayLate
(13)
RentDue
[Now > Time + 30 days]
AwaitingPay
(14)
RentLate
[Now > TimeLate + 5 days]
Conflict
3x
2x
(2)
PayRent
(4) (10)
EndLease
[Now > CreationTime + 12 months]
LeaseEnd
(9)
Extend
[Now > CreationTime + 12 months]
(5)
EarlyTerminate
[Now < CreationTime + 11 months]
Payed-
Extended
(13)
RentDue
[Now > Time + 30 Days]
AwaitingPay-
Extended
(14)
RentLate
[Now > TimeLate + 5 days]
Conflict-
Extended
(3)
PayLate
(2)
PayRent
FinalToBe-
Payed
(6)
GiveNotice
(14)
RentLate
[Now > TimeLate + 5 days]
(7) (8) (11)
TerminateContract
[Now > ConflictTime + 5 days]
(12)
ReturnDeposit
(12)
ReturnDeposit
(15)
EndLease
[Now < CreationTime + 12 months]
Finished
LeaseClosed
(3)
PayLate
[Notice==True]
(7) (8)
TerminateContract
[Now > ConflictTime + 5 days]

ADICO statements

	Actor (A)	Deontic (D)	aIm (I)	Condition (c)	Or else (O)
1	Tenant	must	pay the security deposit		or else the contract is not initiated
2	Tenant	must	pay the rent	every month	or else the rent is late
3	Tenant	must	pay the rent and a fine	if the rent is late	or else the contract can be terminated
4	Tenant	may	end the lease contract	when the original lease period ends	or else the lease contract may be extended
5	Tenant	may	give notice	before the lease period ends by forfeiting his security deposit	
6	Tenant	may	give one-month notice to leave	if the original lease period has ended	
	Landlord	must	launch the contract		
7	Landlord	may	terminate the contract	if the rent is ten days late	
8	Landlord	may	keep the security deposit	if the rent is ten days late	
9	Landlord	may	extend the lease contract	if the lease period ends	or else the lease contract will end
10	Landlord	may	end the lease contract	when oringinal the lease period ends	or else the lease contract will be extended
11	Landlord	may	terminate the contract	if notice has been given and final rent is 10 days late	
12	Landlord	must	return the security deposit to tenant	if the end of the lease has been reached	
13	System	must	shift to accepting payments	every month	
14	System	must	register the rent to be late	if five days have passed since the accepting payment status has been reached	
15	System	must	end the lease contract	if final payment has been made within 12 months	

E. Control Sheets Experiment

Scenario 1:

The landlord (1) has created (2) the contract (3).

The tenant (4) pays (5) security deposit (6) to get to the state paid (7).

The system (8) transitions through transition rentDue (9) to the state Awaitingpay (10) after 30 days (11).

The tenant (12) pays (13) the rent (14) on time (15) to get to the state paid (16). This happens 12 times/ one year (17).

The Landlord (18) extends (19) after 12 months have passed (20) to get to the state PaidExtended (21).

The system (22) transitions through transition rentDue (23) to the state AwaitingPay (24) after 30 days (25).

The tenant (26) pays (27) the rent (28) on time (29) to get to the state paid (30). This happens 3 times (31).

The tenant (32) gives notice (33) to get to the state finaltobepaid (34).

The tenant (35) pays (36) the rent (37) for the final time to get to the state leaseend (38) .

The landlord (39) returns (40) the deposit (41) to finish the contract (42).

Yellow = Actor, Green = Action, Blue = Construct, Purple = Consequence, Red = Condition

Scenario 2:

The landlord (1) has created (2) the contract (3).

The tenant (4) pays (5) a security deposit (6) to get to the state paid (7).

The system (8) transitions through transition rentDue (9) to the state Awaitingpay (10) after 30 days (11).

The tenant (12) pays (13) the rent (14) on time (15) to get to the state paid (16). This happens twice (17).

The tenant (18) has not paid within five days (19), so the system (20) transitions through rentLate (21) to the state conflict (22).

The tenant (23) pays (24) the rent and a fine (25) to get to the state paid (26).

The tenant (27) makes the transition earlyterminate (28) with the contract being younger than 12 months (29) to get to the state finaltobepaid (30).

The tenant (31) fails to pay (32) within five days (33), so the system (34) transitions through RentLate (35) to get to the state ConflictExtended (36).

The Landlord (37) terminates (38) the contract (39) without returning the security deposit (40) to get to the state finished/ to finalize the contract (41).

Yellow = Actor, Green = Action, Blue = Construct, Purple = Consequence, Red = Condition

F. SPSS Output of the Analyses

i. Data output prior knowledge

T-test

Group Statistics

	Preknowledge	N	Mean	Std. Deviation	Std. Error Mean
Score_scenario_1_percentage	Y	8	60,7143	16,73939	5,91827
	N	8	58,6310	19,08744	6,74843
Score_scenario_2_percentage	Y	8	56,4024	19,57470	6,92070
	N	8	61,2805	21,39987	7,56600

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Score_scenario_1_percentage	Equal variances assumed	,022	,885	,232	14	,820	2,08333	8,97592	-17,16811	21,33477
	Equal variances not assumed			,232	13,766	,820	2,08333	8,97592	-17,19892	21,36559
Score_scenario_2_percentage	Equal variances assumed	,083	,777	-,476	14	,642	-4,87805	10,25380	-26,87026	17,11417
	Equal variances not assumed			-,476	13,890	,642	-4,87805	10,25380	-26,88659	17,13049

ii. Data output scenario 1

T-test

Group Statistics

	Version	N	Mean	Std. Deviation	Std. Error Mean
Score_scenario_1	A	8	21,0000	6,96932	2,46403
	B	8	29,1250	5,27629	1,86545
Actor_score_1_percentage	A	8	33,7500	21,33910	7,54451
	B	8	66,2500	14,07886	4,97763
Action_score_1_percentage	A	8	66,2500	18,46812	6,52947
	B	8	73,7500	15,05941	5,32430
Construct_score_1_percentage	A	8	79,1667	21,36233	7,55272
	B	8	91,6667	12,59882	4,45435
Consequence_score_1_percentage	A	8	31,9444	30,53751	10,79664
	B	8	51,3889	30,82493	10,89826
Condition_score_1_percentage	A	8	32,5000	28,15772	9,95526
	B	8	60,0000	30,23716	10,69045

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Score_scenario_1	Equal variances assumed	,380	,547	-2,629	14	,020	-8,12500	3,09052	-14,75351	-1,49649
	Equal variances not assumed			-2,629	13,040	,021	-8,12500	3,09052	-14,79959	-1,45041
Actor_score_1_percentage	Equal variances assumed	1,368	,262	-3,596	14	,003	-32,50000	9,03861	-51,88589	-13,11411
	Equal variances not assumed			-3,596	12,123	,004	-32,50000	9,03861	-52,17124	-12,82876
Action_score_1_percentage	Equal variances assumed	,241	,631	-,890	14	,388	-7,50000	8,42509	-25,57002	10,57002
	Equal variances not assumed			-,890	13,455	,389	-7,50000	8,42509	-25,63893	10,63893
Construct_score_1_percentage	Equal variances assumed	,560	,467	-1,426	14	,176	-12,50000	8,76840	-31,30636	6,30636
	Equal variances not assumed			-1,426	11,344	,181	-12,50000	8,76840	-31,72796	6,72796
Consequence_score_1_percentage	Equal variances assumed	,002	,969	-1,268	14	,226	-19,44444	15,34078	-52,34714	13,45825
	Equal variances not assumed			-1,268	13,999	,226	-19,44444	15,34078	-52,34741	13,45853
Condition_score_1_percentage	Equal variances assumed	,121	,733	-1,883	14	,081	-27,50000	14,60797	-58,83098	3,83098
	Equal variances not assumed			-1,883	13,930	,081	-27,50000	14,60797	-58,84586	3,84586

iii. Data output scenario 2

T-test

Group Statistics

	Version	N	Mean	Std. Deviation	Std. Error Mean
Score_scenario_2_percentage	A	8	46,6463	20,13119	7,11745
	B	8	71,0366	10,46529	3,70004
Actor_score_2_percentage	A	8	35,2273	28,09896	9,93448
	B	8	64,7727	19,70113	6,96540
Action_score_2_percentage	A	8	56,8182	20,47258	7,23815
	B	8	80,6818	11,33112	4,00616
Construct_score_2_percentage	A	8	67,5000	21,21320	7,50000
	B	8	82,5000	12,81740	4,53163
Consequence_score_2_percentage	A	8	29,6875	37,16367	13,13934
	B	8	56,2500	25,87746	9,14906
Condition_score_2_percentage	A	8	45,0000	23,29929	8,23754
	B	8	75,0000	23,29929	8,23754

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Score_scenario_2_percentage	Equal variances assumed	5,482	,035	-3,041	14	,009	-24,39024	8,02174	-41,59518	-7,18531
	Equal variances not assumed			-3,041	10,526	,012	-24,39024	8,02174	-42,14348	-6,63701
Actor_score_2_percentage	Equal variances assumed	2,383	,145	-2,435	14	,029	-29,54545	12,13304	-55,56824	-3,52266
	Equal variances not assumed			-2,435	12,543	,031	-29,54545	12,13304	-55,85476	-3,23615
Action_score_2_percentage	Equal variances assumed	2,605	,129	-2,885	14	,012	-23,86364	8,27285	-41,60714	-6,12013
	Equal variances not assumed			-2,885	10,921	,015	-23,86364	8,27285	-42,08819	-5,63909
Construct_score_2_percentage	Equal variances assumed	3,465	,084	-1,712	14	,109	-15,00000	8,76275	-33,79422	3,79422
	Equal variances not assumed			-1,712	11,510	,114	-15,00000	8,76275	-34,18292	4,18292
Consequence_score_2_percentage	Equal variances assumed	,946	,347	-1,659	14	,119	-26,56250	16,01086	-60,90238	7,77738
	Equal variances not assumed			-1,659	12,496	,122	-26,56250	16,01086	-61,29424	8,16924
Condition_score_2_percentage	Equal variances assumed	,055	,818	-2,575	14	,022	-30,00000	11,64965	-54,98601	-5,01399
	Equal variances not assumed			-2,575	14,000	,022	-30,00000	11,64965	-54,98601	-5,01399

iv. Data output efficiency

T-test

Group Statistics

	Version	N	Mean	Std. Deviation	Std. Error Mean
Time1_seconds	A	8	313,25	102,942	36,396
	B	8	481,00	194,444	68,746
Time2_seconds	A	8	297,13	125,143	44,245
	B	8	396,63	176,823	62,516

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
Time1_seconds	Equal variances assumed	1,318	,270	-2,157	14	,049	-167,750	77,786	-334,584	-,916
	Equal variances not assumed			-2,157	10,638	,055	-167,750	77,786	-339,669	4,169
Time2_seconds	Equal variances assumed	,742	,404	-1,299	14	,215	-99,500	76,589	-263,767	64,767
	Equal variances not assumed			-1,299	12,606	,217	-99,500	76,589	-265,488	66,488

Correlation test

Correlations

		Time1_seconds	Time2_seconds	Score_scenario_1	Score_scenario_2
Time1_seconds	Pearson Correlation	1	,873**	,453	,432
	Sig. (2-tailed)		,000	,078	,094
	N	16	16	16	16
Time2_seconds	Pearson Correlation	,873**	1	,457	,436
	Sig. (2-tailed)	,000		,075	,092
	N	16	16	16	16
Score_scenario_1	Pearson Correlation	,453	,457	1	,901**
	Sig. (2-tailed)	,078	,075		,000
	N	16	16	16	16
Score_scenario_2	Pearson Correlation	,432	,436	,901**	1
	Sig. (2-tailed)	,094	,092	,000	
	N	16	16	16	16

**. Correlation is significant at the 0.01 level (2-tailed).