

Tilburg University

From legal contracts to smart contracts and back again: Towards an automated approach

Butijn, Bert-Jan

DOI:
[10.26116/d6h4-7r79](https://doi.org/10.26116/d6h4-7r79)

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

[Link to publication in Tilburg University Research Portal](#)

Citation for published version (APA):
Butijn, B.-J. (2022). *From legal contracts to smart contracts and back again: Towards an automated approach*. CentER, Center for Economic Research. <https://doi.org/10.26116/d6h4-7r79>

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

From Legal Contracts to Smart Contracts and Back Again: An Automated Approach

BERT-JAN BUTIJN

From Legal Contracts to Smart Contracts and Back Again: An Automated Approach

Proefschrift ter verkrijging van de graad van doctor aan Tilburg University op gezag van de rector magnificus, prof. dr. W.B.H.J. van de Donk, in het openbaar te verdedigen ten overstaan van een door het college voor promoties aangewezen commissie in de Aula van de Universiteit op vrijdag 23 september 2022 om 13.30 uur

door

Berend Johannes Butijn

geboren op 26 augustus 1989 te Leeuwarden.

Promotores:

| | |
|-----------------------------------|---------------------------|
| prof. dr. W.J.A.M. van den Heuvel | Tilburg University & JADS |
| prof. dr. G.M. Duijsters | Tilburg University |

Copromotor:

| | |
|-----------------------|---|
| dr. ir. D.A. Tamburri | Eindhoven University of Technology & JADS |
|-----------------------|---|

Leden promotiecommissie:

| | |
|-------------------------------|-------------------------------|
| prof. dr. S. Tai | Technische Universität Berlin |
| prof. dr. ir. W. van der Valk | Tilburg University |
| prof. dr. E. van Heck | Erasmus University Rotterdam |
| prof. dr. F. Lumineau | HKU Business School |
| prof. dr. M. Mecella | Sapienza University of Rome |

©Berend Johannis Butijn, The Netherlands. All rights reserved. No parts of this thesis may be reproduced, stored in a retrieval system or transmitted in any form or by any means without permission of the author. Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd, in enige vorm of op enige wijze, zonder voorafgaande schriftelijke toestemming van de auteur.

"For a man to conquer himself is the first and noblest of all victories."

– Plato

Abstract

Over the past decade, blockchain technology, a decentralized alternative to the traditional centralized (financial) transaction system, has gained tremendous popularity across the globe and has sparked the revival of smart contracts by providing a platform to support their deployment and execution. Central to the original idea of smart contracts is that every computable clause of a contract is translated into a computer program, and letting the program decide what happens during its life span. The term smart legal contract has been coined to describe smart contracts that aim to capture legally binding agreements between parties.

With some exceptions, works on smart legal contracts assume the smart contract code is the contract. This approach raises the problem, because in most countries contracting parties are required by law to understand the contract. Besides this problem, the aim of letting "smart" legal contracts self-enforce the terms and conditions of a legal contracts introduces a host of additional challenges. Recent efforts have sought to address these challenges; however, they only cater for the deployment of smart contracts for a specific platform. Given the rapid developments in the field of blockchain and smart contracts and the lack of a dominant blockchain platform a platform independent approach is required.

The primary purpose of this research is to develop an understanding of what method would facilitate the creation of smart legal contracts that constitute a legally binding contract and that can (partially) self-enforce their terms and conditions within that contract, regardless of the blockchain platform. Blockchain technology provides the infrastructure to store smart contracts and execute transactions. It is therefore crucial to create an understanding of the inner workings this blockchain technology. The extant literature on BCT reviewed in this dissertation to establish a definition of BCT using a FCA, to delineate the architecture of BCT, point out challenges and characteristics that the technology faces, and provide a roadmap for future research. Another chapter provides insights into the relation between blockchain technology and smart contracts. In this chapter a motivational example, further fleshed out by a case study exemplifies this relation and demonstrates how smart contracts can be employed for business transactions.

In this dissertation a method is presented based on the Model Driven Architecture (MDA) philosophy to (semi) automatically draft smart legal contracts. In line with the MDA philosophy this research presents a domain ontology that facilitates the stipulation of domain specific requirements. The concepts for the domain ontology were identified by reviewing 19 works from the fields of legal requirements engineering, MAS, and e-contracts and conducting a qualitative content analysis. The use of smart legal contracts introduces several problems. Another review was performed on 29 papers to identify these problems. Findings suggest that the identified problems are coupled to distinct parts of a smart contracts' life-cycle and concepts in the domain ontology.

To model smart contracts, a platform specific model (PSM) for the Ethereum platform, and another for the Hyperledger platform are presented. A representation of the concepts required to model smart contracts for both platforms are portrayed in a platform independent model (PIM) that facilitates cross platform usage. Finally, to enable the translation between the models a mapping between the concepts in the domain model and PIM is presented. The models enhance the academic understanding of the smart contract artifact, and its implementation into legal practice.

Acknowledgements

The finalization of this dissertation marks the end of my period as a Ph.D. student at the Jheronimus Academy of Data Science and Tilburg University. My journey as a Ph.D. student would have never commenced without the encouragement and support of Prof. E.W. Berghout for which I would like to express my thanks. During my time as Ph.D. student I got the opportunity to further develop my programming and research skills while enhancing my knowledge on blockchain technology, and IT in general. For this opportunity to develop myself as a person I feel greatly privileged and deeply thankful. The last years have not only been exciting, but also challenging at times. I am thankful for all the support that I received during the research.

First, I would like to express my sincere gratitude to my promotor Prof. W.J.A.M. van den Heuvel. Prof. van den Heuvels' efforts made it possible for me to become a Ph.D. student and for that I owe him great thanks. Furthermore, Prof. van den Heuvel has supported me on both methodological and non-methodological matters, for which I am grateful. The brainstorm sessions that Prof. van den Heuvel and myself engaged in helped to shape my research.

It is a genuine pleasure to express my deep gratitude to my second promotor Prof. G. Duijsters. I would like to thank Prof. Duijsters especially for conveying his enthusiasm for smart contracts and blockchain technology in general, that have led to interesting conversations about the present and future of these promising technologies. His enthusiasm enhanced my motivation when conducting my research and lifted my spirit when needed. Whenever I felt that my work lacked the proper structure Prof. G. Duijsters provided invaluable insights and suggestions that aided in structuring my work.

I owe a deep sense of thanks and gratitude to my copromotor Dr. D.A. Tamburri. Conducting relevant research following a rigorous method is a complex and difficult undertaking. Under the expert guidance of Dr. Tamburri however, I always felt that it became far easier. Not in the last place because I was in the fortunate position to tap into his great expertise on Model Driven Architecture, software engineering, IT architecture in general, and methodological matters. The review process of publishing your research paper can be tedious, and sometimes it seems endless. This burden became significantly lessened because Dr. Tamburri has a great sense of humor that he leveraged frequently to shine a more relaxed perspective on the challenges at hand.

Besides my (co)promotores I would like these acknowledgements to thank the remainder of my thesis committee: Prof. Dr. Tai, Prof. Dr. van der Valk, Prof. Dr. van Heck, Prof. Dr. Lumineau and Prof. Dr. Mecella for their encouragement, insightful comments and challenging questions. The comments and suggestions made by the thesis committee to revise parts of this dissertation have improved the quality of my dissertation, for which I am grateful.

At times writing your dissertation can be frustrating and lonely process. I never felt that I was really alone in my endeavor because I had the support of my friends and family. I would like to especially thank my sisters for their discussions about this research, and my parents for their ongoing support in all things that I do throughout my life. Lastly, I would like to express my deepest thanks to Leontine who strongly supported me and provided me with so much personal care while writing my thesis.

Bert-Jan Butijn
July 20, 2022

Table of Contents

| | |
|--|------------|
| Abstract | ii |
| Acknowledgements | iii |
| 1 Introduction | 1 |
| 1.1 Introduction | 1 |
| 1.2 Motivation | 2 |
| 1.3 Problem Context | 3 |
| 1.3.1 Designing Legally Binding Smart Contracts | 3 |
| 1.3.2 Designing Legal Contract Enforcing Smart Contracts | 5 |
| 1.4 Methods to Use Smart Legal Contracts as Alternative to Legal Contracts . . | 6 |
| 1.5 Research Purpose and Requirements | 9 |
| 1.5.1 Research Purpose and Question | 9 |
| 1.5.2 Requirements | 10 |
| 1.6 Outline of the Dissertation | 11 |
| 2 Blockchain Technology | 14 |
| 2.1 Introduction | 14 |
| 2.1.1 Related Work | 15 |
| 2.2 Background and Basic Notions | 16 |
| 2.2.1 Historical Setting | 16 |
| 2.2.2 Terms and Definitions | 16 |
| 2.3 Research Methodology | 18 |
| 2.3.1 Data Preparation Approach | 18 |
| 2.3.2 Search strategy | 19 |
| 2.3.3 Data Analysis | 21 |
| 2.3.4 Inter-Rater Reliability Assessment | 22 |
| 2.3.5 Sample Selection Results | 22 |
| 2.4 A Systematic Definition of Blockchain Technology | 25 |
| 2.5 Blockchain Technology: Architecture Elements | 26 |
| 2.5.1 Logical View | 27 |
| 2.5.2 Development View | 27 |
| 2.5.3 Process View | 27 |
| 2.5.4 Physical View | 29 |
| 2.6 Blockchain Use-Case View: Main Usage Scenarios | 30 |
| 2.7 Blockchain Technology: Main Architecture Properties | 30 |
| 2.8 Blockchain Technology: Challenges and Outlook | 31 |
| 2.8.1 Latency | 31 |
| 2.8.2 Throughput | 32 |
| 2.8.3 Data Storage | 32 |
| 2.8.4 Data Privacy | 32 |
| 2.8.5 Governance | 33 |
| 2.8.6 Usability | 33 |

| | | |
|----------|--|-----------|
| 2.9 | Discussion | 34 |
| 2.9.1 | A Grounded-Theory of Blockchain Technology | 34 |
| 2.9.2 | Highlights and Observations | 39 |
| 2.10 | Research Gaps and Roadmap | 43 |
| 2.10.1 | Consensus Protocols | 43 |
| 2.10.2 | Data Storage and Privacy | 44 |
| 2.10.3 | Smart Contracts | 44 |
| 2.10.4 | Usability | 44 |
| 2.11 | Limitations and Threats to Validity | 45 |
| 2.12 | Conclusions | 46 |
| 3 | Smart contracts | 47 |
| 3.1 | Introduction | 47 |
| 3.2 | Motivating Example | 48 |
| 3.3 | Lifecycle of Smart Contract Driven Business Transactions | 49 |
| 3.4 | A Reference Architecture for Smart Contract Driven Business Transactions | 50 |
| 3.5 | Motivating Example with Smart Contracts | 52 |
| 3.5.1 | Case Study Design | 53 |
| 3.5.2 | Ethereum Solidity Smart Contracts | 54 |
| 3.6 | Discussion | 55 |
| 3.7 | Conclusion | 56 |
| 4 | Research Methodology | 58 |
| 4.1 | Introduction | 58 |
| 4.2 | Research Paradigm | 59 |
| 4.3 | Model Driven Architecture | 60 |
| 4.3.1 | Modeling Levels | 61 |
| 4.3.2 | Modeling Languages | 62 |
| 4.3.3 | Mappings and Transformations | 64 |
| 4.3.4 | MDA: Benefits and Considerations | 66 |
| 4.4 | Research Design | 67 |
| 4.4.1 | Rationale for the Use of MDA | 67 |
| 4.4.2 | Research Overview | 68 |
| 4.5 | Discussion | 69 |
| 4.6 | Conclusion | 70 |
| 5 | Domain Ontology For Smart Legal Contracts | 72 |
| 5.1 | Introduction | 72 |
| 5.2 | Related Work: Modeling Smart Legal Contracts | 73 |
| 5.3 | Digitizing Legal Contracts | 74 |
| 5.4 | Research Methodology | 75 |
| 5.4.1 | Data Gathering | 76 |
| 5.4.2 | Data Analysis | 76 |
| 5.4.3 | Constructing the ontology | 77 |
| 5.5 | Smart Contracts: Legal Challenges | 77 |
| 5.5.1 | Drafting and Coding | 78 |
| 5.5.2 | Testing | 79 |
| 5.5.3 | Deployment | 79 |
| 5.5.4 | Signing | 79 |
| 5.5.5 | Monitoring and Execution | 80 |
| 5.5.6 | Dispute Resolution and Termination | 80 |

| | | |
|----------|---|------------|
| 5.6 | Motivating Example | 81 |
| 5.7 | A Domain Ontology For Smart Legal Contracts | 82 |
| 5.7.1 | Asset and Party | 82 |
| 5.7.2 | Definition | 83 |
| 5.7.3 | Clause | 84 |
| 5.7.4 | Action | 87 |
| 5.8 | Evaluation: Instantiating the Domain Ontology | 93 |
| 5.9 | Discussion | 100 |
| 5.10 | Conclusion | 102 |
| 6 | Towards a Unified Platform Independent Model for Smart Contracts | 104 |
| 6.1 | Introduction | 104 |
| 6.2 | Related Work: Model Driven Smart Contract Development | 105 |
| 6.3 | Research Methodology | 106 |
| 6.3.1 | Phase 1: Preparation | 107 |
| 6.3.2 | Phase 2: Anchoring | 107 |
| 6.3.3 | Phase 3: Iterative Improvement | 108 |
| 6.4 | Platform Specific Perspectives On Smart Contract Platforms | 110 |
| 6.4.1 | A Platform Specific Perspective of Ethereum | 110 |
| 6.4.2 | A Platform Specific perspective of Hyperledger Fabric blockchains | 115 |
| 6.5 | Towards a Platform Independent Metamodel for Smart Contracts | 121 |
| 6.5.1 | Finding Common Ground | 121 |
| 6.5.2 | A Platform Independent Model for Smart Contracts | 123 |
| 6.5.3 | Transformation Rules | 126 |
| 6.6 | Smart Contract Generation | 128 |
| 6.6.1 | Validation Platform Independent Model | 128 |
| 6.6.2 | Ethereum PSM Creation and Code Generation | 130 |
| 6.6.3 | Hyperledger Fabric PSM Creation and Code Generation | 133 |
| 6.7 | Discussion | 135 |
| 6.8 | Conclusion | 137 |
| 7 | Discussion | 138 |
| 7.1 | Introduction | 138 |
| 7.2 | Design Principles and Lessons Learned | 138 |
| 7.2.1 | A Unified Domain Ontology for Smart Legal Contracts | 139 |
| 7.2.2 | Representation of legal contracts | 142 |
| 7.2.3 | Enforcement with smart legal contracts | 144 |
| 7.2.4 | Achieving Complete Traceability | 147 |
| 7.2.5 | Development of Platform Agnostic Smart Contracts | 147 |
| 7.3 | Limitations and Threats to Validity | 148 |
| 7.3.1 | Internal Validity | 148 |
| 7.3.2 | Construct Validity | 149 |
| 7.3.3 | External validity | 149 |
| 7.4 | A Research Agenda for Smart Legal Contracts | 150 |
| 7.4.1 | Legal Research Opportunities | 150 |
| 7.4.2 | Technical Research Opportunities | 151 |
| 7.4.3 | Business Research Opportunities | 152 |
| 7.5 | Conclusion | 153 |
| 8 | Conclusion | 155 |
| 8.1 | Introduction | 155 |

| | | |
|----------|---|------------|
| 8.2 | Overview of the Research and Main Observations | 155 |
| 8.3 | Theoretical Contributions | 157 |
| 8.4 | Practical Contributions and Implications | 159 |
| A | Research Methodology Blockchain Technology | 161 |
| A.1 | Formal Concept Analysis Methodology | 161 |
| A.2 | Grounded Theory Analysis Method | 161 |
| A.3 | Inter-rater process | 162 |
| B | A 4+1 Architectural View of Blockchain Technology | 164 |
| B.1 | A 4+1 View of Blockchain Technology | 164 |
| B.1.1 | Logical View | 164 |
| B.1.2 | Development View | 166 |
| B.1.3 | Process view | 168 |
| B.1.4 | Physical View | 174 |
| C | Example lease agreement | 177 |
| D | Instantiation of Motivating Example | 182 |
| E | Questionnaire and initial concepts Delphi method | 193 |
| E.1 | Intro | 193 |
| E.2 | Instructions | 193 |
| F | Enum Types Used in Models | 195 |
| F.1 | Enums used for Ethereum smart contract metamodel | 195 |
| F.2 | Enums used for Hyperledger smart contract metamodel | 196 |
| F.3 | Enums used for platform independent model | 196 |
| G | Transformation Rules for Models | 197 |
| G.1 | Transformation Rules From PIM to Ethereum PSM | 197 |
| G.2 | Transformation Rules From PIM to Hyperledger PSM | 202 |
| G.3 | Constraints for the Platform Independent Smart Contract Model | 210 |
| | Bibliography | 215 |

List of Figures

| | | |
|------|---|-----|
| 1.1 | The smart legal contract life cycle | 5 |
| 1.2 | Spectrum of smart legal contracts. Adopted from: [186] | 6 |
| 1.3 | Outline and structure of the dissertation | 13 |
| 2.1 | Research methodology, an outline. | 19 |
| 2.2 | Sample Search and selection strategy, a process model. | 20 |
| 2.3 | Sample results; Grey and Scientific Literature across primary studies. | 23 |
| 2.4 | Sample results; Publication Venues per Year. | 23 |
| 2.5 | Sample results; Topics frequency analysis. | 24 |
| 2.6 | Trends in BCT publication topics. | 25 |
| 2.7 | Distribution topics scientific and grey literature. | 26 |
| 2.8 | Frequency of top 10 most recurring codes. | 34 |
| 2.9 | BCT software elements and properties, an overview. | 35 |
| 2.10 | Consensus protocols related to challenges, in chronological order from left to right. | 36 |
| 2.11 | Trends in BCT challenges from 2008 to 2019. | 39 |
| 2.12 | Decision-making model for blockchain networks; adapted from: [272]. | 43 |
| 3.1 | Supply chain example, Adopted from [265] | 48 |
| 3.2 | Life cycle of smart contract driven business transactions | 49 |
| 3.3 | A reference architecture for smart contract driven business transactions | 51 |
| 3.4 | Motivating example with smart contracts | 53 |
| 3.5 | Sequence diagram illustrating part of a business transaction | 54 |
| 4.1 | Design Science Research Model (DSRM) | 60 |
| 4.2 | Use of different models within MDA. Inspired by [233] | 61 |
| 4.3 | Hierarchy of metamodels and languages. Based on [144]. | 62 |
| 4.4 | Model transformations in MDA. Based on: [104] | 64 |
| 4.5 | Research method in relation to stakeholders and requirements | 68 |
| 4.6 | Overview overall research approach | 69 |
| 5.1 | Composition of a contract | 82 |
| 5.2 | Model elements related to a definition | 83 |
| 5.3 | Model elements related to a clause | 84 |
| 5.4 | Model elements related to an equations | 85 |
| 5.5 | Relation between actions, resources, agents and terms. | 88 |
| 5.6 | Contract level concepts displayed in Contract Custodian | 99 |
| 5.7 | Action level concepts displayed in Contract Custodian | 100 |
| 6.1 | Delphi methodology. Adopted from [114] | 107 |
| 6.2 | A model of Ethereum blockchain architecture | 110 |
| 6.3 | A metamodel of Ethereum smart contracts | 113 |
| 6.4 | A model of the Hyperledger Fabric blockchain architecture | 116 |
| 6.5 | An overview of resources for Hyperledger Fabric | 118 |

| | | |
|------|---|-----|
| 6.6 | A Metamodel of Hyperledger Fabric smart contracts | 119 |
| 6.7 | A platform independent model for smart contracts | 123 |
| 6.8 | Algorithmic steps for Smart Contract generation | 128 |
| 6.9 | Activities of algorithm to create Ethereum smart contracts | 130 |
| 6.10 | Activities of algorithm to create Hyperledger smart contracts | 134 |
| F.1 | Enum types used for Ethereum smart contract metamodel | 195 |
| F.2 | Enum types used for Hyperledger Fabric smart contract metamodel | 196 |
| F.3 | Enum types used for PIM smart contract metamodel | 196 |

List of Tables

| | | |
|-----|--|-----|
| 2.1 | Quality criteria grey literature. | 21 |
| 2.2 | A complete reference over consensus protocols; instances are described along their characteristics, an implementation example and the source for argument of the claims. | 40 |
| 5.1 | Overview of Legal Challenges Related to Smart Contracts | 78 |
| 5.2 | Liabilities and privileges by party role. | 90 |
| 6.1 | Descriptive statistics respondents | 109 |

List of Abbreviations

| | | |
|--------------------|--|------------|
| BCT | Block Chain Technology | 1 |
| MDA | Model Driven Architecture | 2 |
| EDI | Electronic Data Interchange | 3 |
| P2P | peer - to - peer | 14 |
| DLT | distributed ledger technology | 14 |
| IoT | Internet of Things | 15 |
| MAS | Multi - Agent Systems | 15 |
| POW | Proof - of - Work | 17 |
| PoS | Proof - of - Stake | 17 |
| DPoS | Delegated - Proof - of - Stake | 17 |
| PoET | Proof - of - Elapsed - Time | 17 |
| ZKP | Zero Knowledge Proofs | 17 |
| PBFT | Practical Byzantine Fault Tolerance | 18 |
| MLR | Multivocal Literature Review | 18 |
| GL | Grey Literature | 18 |
| SL | scientific literature | 19 |
| FCA | Formal Concept Analysis | 21 |
| GT | Grounded Theory | 22 |
| DApps | Decentralized Applications | 27 |
| GHOST | Greedy Heaviest Observed Subtree | 31 |
| GDPR | General Data Protection Regulation | 33 |
| QoS | Quality of Service | 50 |
| SLAs | Service Level Agreements | 50 |
| SOA | Service - Oriented Architecture | 50 |
| DSR | Design Science Research | 58 |
| DSRM | Design Science Research Model | 59 |
| CIM | Computation Independent Model | 61 |
| PIM | Platform Independent Model | 61 |
| PSM | Platform Specific Model | 61 |
| MOF | Meta Object Facility | 62 |
| UML | Universal Modeling Language | 62 |
| DSL | Domain - Specific Language | 63 |
| BPMN | Business Process Model Notation | 63 |
| OCL | Object Constraint Language | 63 |
| ATL | Atlas Transformation Language | 64 |
| e-contracts | electronic contracts | 74 |
| EOA | Externally Owned Accounts | 111 |
| NFT | Non - Fungible Token | 111 |
| EVM | Ethereum Virtual Machine | 111 |
| CA | Certificate Authority | 116 |

*Dedicated to my family and friends for supporting me, and God. Lord
knows how hard I needed Him.*

Chapter 1

Introduction

1.1 Introduction

Smart contracts, programs with the potential to automate transactions and beyond, have gained tremendous popularity over the past years. Central to the original idea by Nick Szabo [245] of smart contracts is that every computable clause of a contract or agreement is encoded into arbitrary computer logic with the aim of coding this logic into computer programs, and let the program decide and execute what happens during the contract's life span. These principles constitute automated and self-enforcing agreements expressed in code. Initially however, the lack of means to digitally exchange assets in safe manner hampered the widespread adoption of smart contracts. The concept of smart contracts saw its revival with the introduction of Block Chain Technology (BCT) [83].

Originally designed for keeping a financial ledger to record transactions, the blockchain paradigm offers the potential to be employed as a technology to underpin smart contracts. The popularity of BCT can mainly be attributed to the fact that it removes the reliance on a centralized authority to facilitate or mediate transactions in any way [265]. BCT is a specific form of distributed ledger technology where the ledger is deployed on a Peer-to-Peer network on which transaction data is replicated, shared, and synchronously distributed. Transactions are processed following a strict consensus protocol that is operated by specific nodes to ensure the validity of requested transactions, and to synchronize all shared copies of the distributed ledger. Where BCT complements smart contracts is that the former enables secure peer-to-peer transactions that are recorded on a tamper proof ledger shared which is auditable to parties concerned with the smart contract [274]. From blockchain technology smart contracts also inherit some important characteristics; Since the smart contracts can be stored on a blockchain they become *immutable* meaning that their contents can not be changed once deployed in order to guarantee that parties cannot tamper with the code. The outcomes of the contract are required to be *deterministic* because every node on the blockchain network needs to be able to replicate the execution of the smart contract and have the same outcome [164].

Smart contract encompasses the terms "smart" and "contract". The term "smart" is derived from the fact that because a smart contract contains pre-defined programming logic it can automatically execute the terms stipulated in the contract [83]. Luu et al. [164] and Cruz, Kaji, and Naoto [63] argue that as such a smart contract can be regarded as an autonomous agent. The second term "contract" suggests that a smart contract will self-enforce the obligations and exercise the rights that are stipulated in the contract [58], which may include seizing control of assets that are stored on a shared ledger, or other means to enforce contractual agreements [83]. Although the term smart contracts includes the word "contract" that does not necessarily imply that they are legally enforceable [56]. The term smart legal contract

has been coined to describe an application of a smart contract that constitutes to a legally binding contract that can (partially) self-enforce the terms and conditions stipulated within that contract.

This research seeks to investigate what method would allow the creation of smart legal contracts that constitute to a legally binding contract and that can (partially) self-enforce their terms and conditions within that contract, regardless of the blockchain platform. The method is developed following the Model Driven Architecture (MDA) philosophy for software design which suits the purpose of this research. In line with the MDA philosophy this research presents a domain ontology that facilitates the stipulation of domain specific requirements. A platform independent model for blockchain smart contracts, and platform specific models to instantiate smart contracts on a Hyperledger blockchain or the Ethereum platform. Combined, these models can be employed to develop smart legal contracts.

In the remainder of this chapter we will first motivate the importance of smart legal contracts to society and why the theoretical development of the concept is warranted. Thereafter in Section 1.3 we will discuss the legal considerations that using smart contracts to substitute legal contracts introduces. In the next section (Section 1.4), recent endeavours to address these challenges will be presented. We conclude the chapter by explaining the purpose of the research and formulating the coherent research question in Section 1.5. Lastly, an outline of the other chapters in this dissertation is provided in Section 1.6.

1.2 Motivation

Nowadays between \$850 and \$930 billion is spend online in the U.S. alone in commercial transactions, and this number is only increasing [256]. Coincidental with this trend, consumers and organizations increasingly transact in a digital manner. Unfortunately, with any economic exchange consumers and organizations incur transaction costs. Transaction costs are costs related to search, measurement, bargaining, and enforcement of an economic exchange. Traditionally, legal contracts are used to govern opportunistic behaviour or lack of adaptation in order to minimize transaction costs [224]. On a global marketplace where the transactions and agreements are made in an online setting traditional contracts no longer suffice. Acknowledging these problems, various countries have adopted laws that allow parties to engage in digital contracting [92].

Smart contracts are able to perform transactions without a trusted third party. Moreover, researchers [285] and institutions [100] argue that smart contracts could greatly reduce transaction costs by lowering enforcement costs; Smart contracts reduce the chance of breach through self-execution and immutability, thereby reducing the need for expensive third-party monitoring or litigation. In concert, automated transaction execution by the smart contract reduces costs and increases the pace of monitoring and verification. Lastly, the use of BCT as underpinning technology for smart contracts provides all parties involved in the agreement with a transparent record of bilateral facts and their evolution, allowing for monitoring without requiring costly replication.

With the increase of digitization and globalization, the call within society for safe and sound manners to engage in electronic contracting grows ever stronger. Smart contracts can be an invaluable technology to address this need. However, at the moment of writing most smart contracts cannot be considered as contracts in the true sense of the word. Rather, they can be perceived as automated escrow handlers that capture simple commitments of parties [186]. This impedes smart contracts from reaching their true potential in enabling more sophisticated, safe, logic based, and legally binding transactions.

The desire to employ digital means for e-commerce is not new. Already in the 1960's the Electronic Data Interchange (EDI) method was pioneered. The EDI method caters for the exchange of structured messages between firms in a format that can be processed by IT. However, organizations averted from adopting EDI due to high installation costs, lack of a unified standard leading to incompatible protocols, and other technological limitations. With the dawn of the internet organizations were provided with an affordable, unified, and flexible manner to exchange messages or other information [54]. It is against this backdrop of digital transformation that Nick Szabo [245] proposed the concept of smart contracts as a method to digitize legal contracts so that IT could process the execution and monitoring of legal agreements. Initially smart contracts enjoyed little success due to a lack of a platform to store the contract and exchange assets [83].

BCT is a promising technology to exchange value and store smart contracts. Moreover, the technology provides several attractive properties that smart contract inherit. Not surprisingly, nowadays a myriad of platforms exist that share the commonality of employing BCT to make transactions and store smart contracts. On the flip side, smart contracts also inherit several technological constraints from blockchain that impede the use of smart contracts to substitute legal contracts. Furthermore, the diversity of BCT platforms has led to a multi-form implementation of the smart contract concept. Potential users of smart legal contracts would therefore be required to inquire into, and develop smart contracts for each platform, making it a costly endeavour. Moreover, the current lack of understanding among scholars of commonalities between platforms hampers the further scientific development of smart contracts.

BCT and smart contracts will have a radical impact on how organizations in the future will govern their relations [161], and are touted as a revolution. This research is warranted as an understanding the needs to be created on these technologies an contractual practice can technically conflate, and the limitations thereof. With organizations becoming evermore reliant on digital commerce for their revenue this need becomes even more urgent.

1.3 Problem Context

A smart legal contract is a specific application of a blockchain based smart contract that constitutes a legally binding contract and that can (partially) self-enforce the terms and conditions stipulated within that contract. This means that the smart contract is used within the context of the legal domain, and more specifically legal contracts. Designing a smart contract in a legally binding manner while enabling them to self-enforce the terms and conditions in the contract introduces several challenges. First we will discuss what types of legal contracts are there and what makes them legally binding as it likely that smart contracts are required to adhere to the same principles [9, 16, 207]. Thereafter we will discuss what additional challenges arise when letting smart contracts self-enforce a contract.

1.3.1 Designing Legally Binding Smart Contracts

The first notion of contracts can trace it's origins back to classical Rome [207]. There are several descriptions of what a legal contract is and how it can be defined¹, and there is no universally accepted definition. For reasons of popularity and clarity we adopt the definition

¹The American Law Institute's Restatement Second of the Law of Contracts defines a contract as "a contract is a promise or a set of promises for the breach of which the law gives a remedy, or the performance of which the law in some way recognizes as a duty." Reinecke et al. [216] provide the following definition: "A contract is a legally enforceable agreement, in which two or more parties commit to certain obligations in return for certain rights."

of legal contracts by Treitel [251]. Treitel defines a legal contract as: "an agreement giving rise to obligations which are enforced or recognized by law". These obligations arise from the mutually agreed upon set of promises of performance that the parties have made towards one another. Depending on the content of the contract, it may be characterized as bilateral or unilateral. When the contract is bilateral, the parties have mutually made a promise or a set of promises to each other. Unilateral contracts are rare instances of agreements where one party makes a (set of) promises and the other parties do not.

Whether a legal contract is legally enforceable by law depends on if it is consistent with the requirements of the law. Within the current legal frameworks a contract needs to meet certain requirements to be legally binding. As de Filippi and Wright [66] note, even in cases where a smart contract might completely substitute a legal contract, these programs do not operate in a legal vacuum. It is likely that smart contracts will need to satisfy the same criteria [9, 16, 207]. A smart contract which does not meet these criteria, does not constitute a legally binding contract, and therefore the contracting parties will not be able to legally enforce its content. Considering that smart contracts might not be able to completely self-enforce all terms and conditions within a legal contract [214] this most likely will be required [170].

Although legal systems differ between countries, they show considerable common ground with regard to the regulation of contracts and the requirements for a legal contract to become binding. The lion's share of codifications (civil and common law) include the principle of freedom of contract, allowing parties to enter into a contract and determining its content. In general, to be legally binding a contract requires the elements of an offer by a party and the acceptance of that offer by another party. Common law systems also require the element of consideration, meaning that something of value is offered by a party to the other [9, 16]. An offer is an expression by a party of willingness to contract on certain conditions, made with the intention that the offer is to be legally bound upon acceptance by another party [251]. An important element of acceptance is that both parties entering into contract *understand the offer* that is being accepted. Acceptance of the offer also needs to be communicated by the accepting party. The communication about the acceptance can take place in several manners: Firstly, implicit when one of the parties conducts an action that implies the intention of a deal (e.g. asking a cab to drive somewhere). Secondly, there is the tacit manner when there is a change to the general terms and conditions of a legal contract by a party and no action is taken by another party within a given period. Finally, there is the explicit manner where there is a written contract, electronic message or oral consent. Written contracts are colloquially referred to as a legal contract in everyday life. In this dissertation, the terms 'legal agreement' and 'legal contract' are used to refer to the terms and conditions of the bilateral legal agreement between the parties as they are set down in writing in the body of the written agreement.

Traditionally, setting down the terms of a legal contract in writing on behalf of clients is the task of legal professionals (e.g. a lawyer). Legal professionals gather any terms and conditions that are relevant for the drafting of the legal contract by deliberating with the business party they represent. Programmers can use the terms and conditions defined within the legal contract as requirements to code a smart contract in a manner that reflects the intent of the contracting parties. Consequently, programmers become an important third stakeholder in a contracting process as they transform the legal requirements into code. However, most legal professionals will not be able to read code and thus verify whether the smart contract code reflects the intent of their clients. Conversely, most programmers will not have a full understanding of the legal domain and the concepts used therein. This lack of a *common understanding* about what concepts are used within the legal domain and how they are translated to code hampers the further use of smart contracts for the legal domain.

1.3.2 Designing Legal Contract Enforcing Smart Contracts

When a smart contract meets the requirements of offer, acceptance, and consideration it constitutes a legally binding contract. However, as discussed, an important prerequisite is that the parties are able to understand what the offer means. It is unlikely that smart contract code alone will be enough for parties entering into contract to understand the offer [64, 76, 80, 90, 179]. Therefore, for this research we assume that a smart contract co-exists with a legal contract equivalent upon which it is based. The second aim of designing a smart contract as a smart "legal" contract is that it can self-enforce the terms and conditions within that legal contract. This second aim introduces a host of additional challenges. A review of the literature reveals that there are 21 challenges to consider when aiming to let a smart contract enforce the terms and conditions within a legal contracts. We will further expound and clarify the results of this review in Chapter 5. Each of these challenges is related to a distinct phase of the life-cycle of a smart contract that will now first be briefly explained (see Fig. 1.1). The lifecycle presented here roughly coincides with that presented by Governatori et al. [96] with the exception of the negotiation and formation that we did not include because it is beyond the scope of this work. Besides this difference, we argue that there is also a testing and signing phase while we conjoined the dispute resolution and termination phase because our analysis shows that these phases have a strong relation.

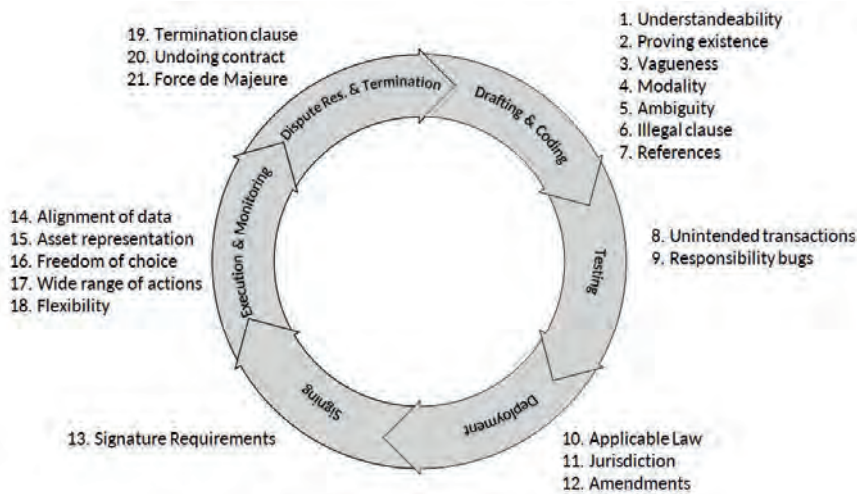


Figure 1.1: The smart legal contract life cycle

In the drafting and coding phase the legal contract and its digital equivalent are "written" by the contractual parties. A legal contract can be written first and thereafter a coded version and vice versa. After the contract has been written in the next phase it needs to be determined that the smart contract does what it is supposed to do and nothing more or less by testing the code. In a next phase the compiled or coded version of the smart contract can be deployed on the blockchain platform. Once the smart contract has been drafted, it can be signed by both parties to confirm their acceptance to the agreement. In the execution and monitoring phase the smart contract monitors the execution of the agreement by the parties and undertakes action itself when stipulated to do so. When dispute resolution between the parties is required, the smart contract enters a new phase that may entail its termination.

Some efforts have been made to overcome these challenges that will be discussed in the section hereafter.

1.4 Methods to Use Smart Legal Contracts as Alternative to Legal Contracts

When Szabo [245] popularized the term smart contract in 1997 he defined it as: "a set of promises, specified in digital form, including protocols within which the parties perform on these promises.". Nowadays smart contracts are predominantly used for business transactions other than legal agreements. Clearly, the present-day popular use of BCT based smart contracts strongly deviates from what Nick Szabo envisioned the concept for, leading to a polysemous understanding of the concept. Since the revival of the concept other definitions have been suggested (e.g by Clack, Bakshi, and Braine [58] and the Accord Project) that are more aligned with the original idea as pioneered by Szabo. Hitherto, there is no definition of smart contracts that has been universally accepted [16, 179, 236].

Despite the recent attempts to redefine smart contracts Stark [242] has argued that the current name smart contract is still imperfect, misleading, and outdated. He argues that the term smart contract sometimes refers to a specific technology, whereas at times the term describes a manner of employing the technology to complement, or replace existing legal agreements. The problem therefore, he posits, is that a smart contract nowadays can be perceived from both a legal and a technical perspective but no distinction is made when the term is used. Stark proposes the term "smart legal contract" to designate smart contracts with the specific purpose of complementing or substituting legal contracts. For this research we adopt the term smart legal contracts and define it as an application of a smart contract that constitutes to a legally binding contract that can (partially) self-enforce the terms and conditions stipulated within that contract.

The term smart legal contracts can be regarded as an umbrella term for various approaches and initiatives with the shared aim of using smart contracts as an alternative to legal contracts. Indeed, there exists a wide spectrum of approaches to create smart legal contracts. In Figure 1.2 this spectrum is depicted. Hereafter each smart legal contract category will be explained, and the works that fit into the respective category reviewed to delineate the full range of design patterns for smart legal contracts.

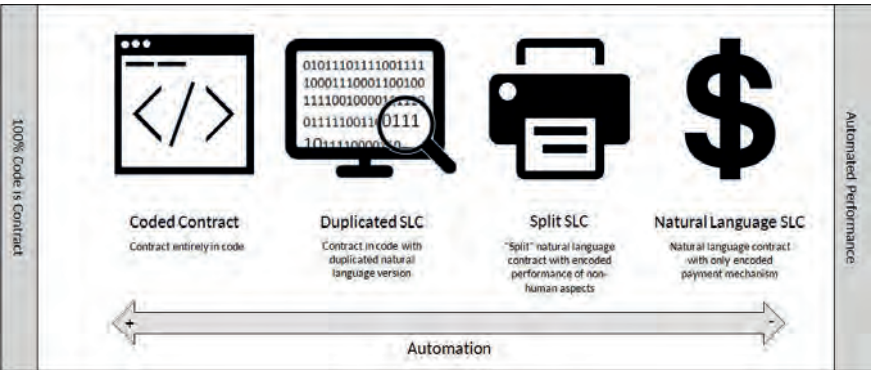


Figure 1.2: Spectrum of smart legal contracts. Adopted from: [186]

Starting from the left of Figure 1.2 are coded smart legal contracts. Approaches to cater for smart legal contracts that fall into this category are not coupled to a legal contract in natural language. Parties using this type of smart contract regard the code as the contract [105]. The more commonly known "normal" smart contracts would fit this approach to a smart legal contract. Since there is no legal contract the code itself stipulates the contract between the parties. Although research on the topic of blockchain based smart contracts remains limited [5] capturing the rules and semantics stipulated in a traditional legal contract to automatically execute these using code is not a novelty. In most countries this would not be a viable solution as parties are required to be able to understand the contract, which is often not the case for coded contracts. To remedy this problem some researchers have introduced a domain specific language (DSL) for smart legal contract development. He et al. [110] introduces SPECS a specification language for smart contracts that compiles to Solidity for Ethereum smart contracts, and further extend the work to better cater for different asset types [286]. Similarly, Wöhrer and Zdun [273] demonstrate the use of a domain specific language for smart contract development. Although these methods provide invaluable efforts to ease the development process of smart contracts, the languages discussed only here only target the Solidity language and thus cannot be reused for other platforms. DSLs are commonly used to capture and stipulate domain specific knowledge. However, it is still a programming language that might be hard to understand for legal practitioners.

At the opposite end of the spectrum there are natural language smart legal contracts that only execute small pieces of a legal contract that concern payments. Contrary to coded contracts, natural language smart legal contracts are coupled to a legal contract. A prime example of this approach is that used for Ricardian contracts [99]. A Ricardian contract is an agreement model to capture the intentions of a contract before it is enacted. In a Ricardian contract hashes are used to refer back and forth to (digital) documents and code. Because a human readable legal contract is available the agreement between the parties can be red. The underlying legal documents can be accessed by the parties at any time as they are hash-stored on a blockchain. Using hashes the code of the Ricardian contract is coupled to clauses in the legal contract. It is important to note that for Ricardian contracts first a legal contract in natural language is written and thereafter the parts that can be automated are coded. A Ricardian contract also includes hidden signatures of the parties concerned with the contract. It is important to note that for these type of contracts automation is limited [186]. Only payment instructions and those for signing of the (digital) documents are encoded. This sacrifices one of the attractive prospects of smart contracts; namely that the contracting process can be automated.

In between these dichotomous types of smart contracts there are two other types of approaches that can be regarded as a hybrid. A core feature of these approaches is that a *digital twin* in the form of a smart contract co-exists with a legal contract. The first of these approaches is where a legal contract in natural language exists with a one-on-one *duplicated smart legal contract* equivalent in code. In essence, for each clause in the legal contract there is matching code that executes this clause and vice versa. The legal contract subsumes the coded version of the contract. In other words, *de jure* the legal contract governs the agreements between the parties, *de facto* the coded version of the smart contract actually enforces the agreement. This has an important implication; if for any reason the technical execution deviates from what the contractual parties agreed upon the legal contract is leading in resolving the issue at hand. Clearly, this needs to be known by the contractual parties in advance. Another, equally important ramification is that the legal contract is required to be *translated* to a coded equivalent that needs to be an exact duplicate. Frantz and Nowostawski [82] present a mapping that they operationalize using a domain-specific language in order to support the contract modeling process. The statements are constructed

from different components abbreviated as ADICO that include: **A**tttributes - describing the actors' attributes, **D**eontic - describing the nature of a statement as deontic logic, **A**lm - describes the outcome that the statement regulates, **C**onditions - describing contextual conditions, **O**r else - describes the consequences of non-conformance. It remains unclear how exactly the ADICO concepts are translated to smart contract executable code. Furthermore, while an interesting approach to smart contract development an empirical evaluation of the proposed method is lacking.

Another example of a hybrid approach are *split smart legal contracts* where non-human performance is encoded in code while other obligations are written in natural language. These two components work seemingly together. The split smart legal contract approach is mostly template based; users fill in the required information in pre-defined templates to create smart contracts and a paper version in natural language. Recently there have been several attempts to design smart legal contract using this approach; In their whitepaper Norte et al. [190] introduce the Angrello framework for legal contracts that employs ontologies and templates to create smart legal contracts. In two whitepapers Clack et al. [56, 58] suggest smart legal contracts based on templates and outline their foundations. Rigorous (replicable) empirical verification and validation by testing these approaches is lacking however. There have not only been scholarly attempts to design approaches that cater for smart legal contracts. Practitioners have also made notable efforts to bridge the gap between legal contracts and smart contracts.

Monax² and OpenLaw³ for instance, offers a method to deploy and manage smart legal contracts on the Ethereum blockchain. Markup language enables the visualization of the user defined template inputs. The stipulation of the contracts in terms of possible action and so on remains limited. Moreover, models that underpin the smart legal contracts are not explicit, making traceability between inputs and output difficult. The templates of Accord⁴ have more relaxed templates that allow users to more freely stipulate clauses in code that can be deployed on a Hyperledger Fabric blockchain. However, this requires more codified input from its users. Consequently, the business logic for the smart contract would need to be coded by a developer in collaboration with a legal professional. Al Khalil et al. [3] argue that therefore more attention should be drawn towards facilitating traditional developers of contracts, namely lawyers and other legal professionals. Despite the fact that the accord project templates offers more discretion when it comes to drawing contracts, users still need to use either predefined contracts or clauses. For some situations more fine-grained stipulation of actions might be required.

While these endeavors are invaluable to foster the widespread employment of smart contracts to substitute or complement traditional contracts between parties, some of their limitations should be mentioned: Firstly, with the exception of CommonAccord⁵ project, the aforementioned projects are exclusively based on either the Ethereum platform or the Hyperledger platform. However, the concept of blockchain based smart contracts is still in its infancy as the field is rapidly developing. Because of this high volatility, this trend could quickly shift towards another platform and programming language. The most obvious implication of this shortcoming is that more rigorous, generalizable, and formalized approach that is *platform agnostic* to create smart legal contracts is missing [2, 59, 217, 253, 288]. Having to rewrite code for each platform is a time consuming and arduous process which greatly hampers broad adoption of smart contracts. Secondly, and perhaps more important, the

²www.monax.io

³www.openlaw.io

⁴www.accordproject.org

⁵<http://www.commonaccord.org/>

methods that underpin the transformation of the annotated legal contract to smart contract code are not explicit. Thus, making *traceability* of the translation process from the legal specification to code impossible. A consequence of this lack of transparency is that contracting parties again have to trust third parties that their contract is translated in line with what they have specified [288]. This is in stark contrast to one of the main advantages of blockchain based smart contracts, namely their potential to capture and execute transaction logic without the need for a trusted third party [128].

1.5 Research Purpose and Requirements

Building on the foundations of the previous sections, this section describes the purpose of the research and coherent research question. Furthermore, it will discuss the requirements of the artefact that is developed for this research.

1.5.1 Research Purpose and Question

The review in the prior section (section 1.4) shows that there exist a wide range of approaches have been undertaken to design smart legal contracts. To avoid any confusion, we deem it appropriate to clarify how we define a smart legal contract. For this research a smart legal contract will be perceived as a specific application of a smart contract that is legally binding, and can (partially) self-enforce conditions of the legal contract upon which it is based and co-exists with.

The primary purpose of this research is to develop a deeper understanding of what *method* would facilitate the development of smart legal contracts that are legally binding and can (partially) self-enforce conditions within a legal contract, regardless of the blockchain platform. Rather than focusing on the development of novel theory or the verification thereof, this research seeks to design an artefact (method) to solve this problem. The aim of the research is *not* to investigate how smart contracts could be technically improved or their security enhanced. Though we acknowledge that testing is an important part of the life cycle of a smart contract we posit that this is beyond the scope of this research, as we are primary focused on how legal concepts can be represented, mapped and how they can be enforced. Moreover, there is a considerable body of literature that has investigated how smart contracts can be tested to ensure safety and to verify behaviour. Tools to develop smart contracts generally remain scarce however [253, 288].

To attain the overall purpose of this research, the following research question has been formulated: "How can smart legal contracts be developed in a manner that constitutes to a legally binding contract, and that can enforce their terms and conditions within that contract, regardless of the blockchain platform". In answering this research question several knowledge questions need to be addressed first. Smart legal contracts are a specific application of smart contracts that are deployed and run on a blockchain. Understanding how BCT and smart contracts relate to each other is therefore a prerequisite to fully grapple the technical (im)possibilities of a employing smart contracts to substitute legal contracts. Therefore, a sub aim of this dissertation is to create this understanding.

From the perspective of the legal domain a common understanding needs to be created about what concepts are used by legal professionals when writing contracts. Identifying these concepts aids in making knowledge about a specific domain explicit. The purpose of making this domain knowledge explicit is to allow legal professionals to stipulate the requirements for their smart legal contract using concepts that are understood within their

profession. Programmers, in turn can employ this knowledge as a specification for the translation of a legal contract. Effectively this gathered domain knowledge is utilized to create a *lingua franca* for legal professionals and programmers on smart legal contracts.

However, as will be discussed in Chapter 5, Section 5.5 there are several challenges when letting smart legal contracts enforce the terms and conditions in a legal contract. Some of these challenges are directly related to the concepts used in the legal domain, or the manner in which legal contracts are drafted. Identifying these challenges and fleshing out to which legal concepts they appertain, aids in predicting, addressing and specifying potential enforcement issues during the life cycle of a smart legal contract. Moreover, nowadays several blockchain platforms exist that allow for the use and deployment of smart contracts. Each of these platforms has its own technical peculiarities and instantiates of the smart contracts concept differently. If the method designed for this research is to cater for the development of smart legal contracts on several platforms, the peculiarities of and commonalities related to the implementation need to be known. Enhancing this knowledge is another sub purpose of this research.

1.5.2 Requirements

In the previous paragraph we stated the purpose of this research, which is to develop a method (artefact) to design smart legal contracts. A specification of what an artefact must be capable of is commonly formulated with requirements [39]. Requirements are conditions or the capability of a system to address a problem or achieve an objective [13]. Gorschek and Wohlin [95] discern various abstraction levels of requirements. For this research we utilise feature level requirements that define which features the method should support without going into detail about what functions are required. This level of abstraction is warranted because we aim to provide stakeholder-driven overview. The purpose of formulating requirements is to define a specification of an artifact that meets the stake holders' needs [213].

Given that the problems identified for this research stem from legal and technical stakeholders, a solution requires to be satisfactory from both the legal and technical the perspective. Firstly, there are the legal aspects to consider to make smart legal contracts truly a "contract" in the legal sense (see Section 1.3). An important ramification of employing smart legal contracts to partially substitute legal contracts is that the method needs to be understandable by programmers and legal professionals as they will need to cooperate. At the moment here are few tools available for domain specific purposes [59, 288] and even fewer for the development of smart legal contracts. Therefore, we include the following requirement:

Requirement 1: The method needs to cater for a manner to make domain concepts explicit so that legal professionals can stipulate the requirements for their smart legal contract.

In Section 1.3.2 we discussed a life cycle model for smart contracts, and the problems that are related to smart legal contracts. Several studies (e.g., [76, 80, 156, 223]) suggest that programmers and legal professionals will need to face several problems during the coding and drafting phase of a smart legal contract (see Section 1.3.2). These problems will be further discussed in Chapter 5. Arguing that these problems need to be anticipated we define the following requirement:

Requirement 2: The method needs to cater for the specification, and potential resolution of issues when drafting and coding.

There are several challenges to be addressed when aiming to make a smart legal contract "smart" and able to self-enforce the conditions stipulated in the legal contract (see Section

1.3.2). The identified challenges are suggested in various sources (e.g., [64, 106, 179, 217, 238, 270]). To address these problems, the following requirement have been formulated:

Requirement 3: The method needs to cater for the specification, and potential resolution of issues related to enforcement of terms and conditions by a smart contract.

How the transformation from the legal contract to the smart contracts takes place is currently not transparent for most existing approaches to develop smart legal contracts. This introduces verifiability issues that lead to trust issues (see Section 1.2). In general, most approaches to develop smart contracts lack verifiability [288]. Therefore, we define the following requirement:

Requirement 4: The method needs to cater for traceability between the specification of the legal contract to the smart contract that will embody this specification.

From a technical perspective, current efforts that aim to support smart legal contract development are geared towards one platform (see Section 1.2). Taking a broader perspective, the same holds true for smart contract developing methods in general [2, 59, 217, 253, 288]. Establishing a platform agnostic perspective of smart contracts will enable the design of smart legal contracts that adhere to the technical principles of multiple blockchain platforms. Thus, it is an aim of this research to flesh out the method in a manner that facilitates the platform agnostic development of smart legal contracts. The following requirement is therefore stipulated:

Requirement 5: The method needs to cater for the creation of smart contracts that are platform agnostic to facilitate cross platform usage.

At the same time, the smart contract ultimately needs to be deployed on a specific platform which requires the method to be applicable to a specific platform. Currently, there are few methods to cater for the development of smart contracts and even less models that capture the concepts of a specific platform [2]. Thus, we define the following requirement:

Requirement 6: The method needs to cater for the creation of smart contracts on a specific platform.

1.6 Outline of the Dissertation

This dissertation is organized into 8 chapters. A context of this research has been delineated in this chapter, along with its purpose and the requirements of the method the research seeks to develop.

Smart contracts are deployed and run on a blockchain infrastructure. Understanding how BCT works is pivotal to grapple the ramifications of this choice for smart contracts. Chapter 2 presents an overview of the literature on BCT, in particular on the trends within the field. More important the chapter delineates the architectural perspectives on the technology, and how it has evolved. Related to the aforementioned, the properties and inherent characteristics of the technology are discussed. Finally, the chapter points out the current challenges for the technology and gaps in literature.

In the following Chapter 3, a background on smart contracts will be provided. Using a motivating example the chapter will firstly delineate how smart contract driven business transactions take place. A generic meta model of most elements related to smart contract driven transactions is provided, along with a reference architecture.

Chapter 4 expounds our research methodology. In the chapter we firstly discuss the main research paradigm for this research, design science. The design of this study is based on the MDA philosophy. An overview of the basic notions of Model Driven Architecture is provided in the chapter. We further explain how software is designed using this approach and what advantages it provides over other methods. Following, we discuss the rationale for adopting MDA as the philosophy underpinning the design of the method. An overview of the research approach followed for this dissertation is presented thereafter.

Following, in Chapter 5 a domain model to model smart legal contracts is presented. The purpose of this chapter is to demonstrate how the contents of a legal contract could be captured in a model. A motivating example is utilized to delineate the relation among the elements of the domain model.

Building on the background provided in Chapters 2 and 3, Chapter 6 is devoted to describing the models that can be employed to write smart contracts. Firstly, the platform specific models for the Ethereum and Hyperledger Fabric platforms are explained. Based on the commonalities between these two platforms a platform agnostic model for blockchain technology that allows for the creation of platform independent smart contracts is presented thereafter.

Chapter 7 discusses how the main research question is addressed. Taking together the findings of constructing the blockchain related models and the smart legal model, the outcomes of the design process are discussed. This is followed by Section 7.3 that addresses the limitations and threats to validity of the study. Derived from the insights of the discussion and the execution of the research we suggest some opportunities for future research in Section 7.4 of Chapter 7.

Finally, Chapter 8 concludes the dissertation. The chapter will first give an overview of the complete research process. After describing the research process the contributions are presented. A summary of the content of the dissertation and the relation between the chapters is depicted in Figure 1.3

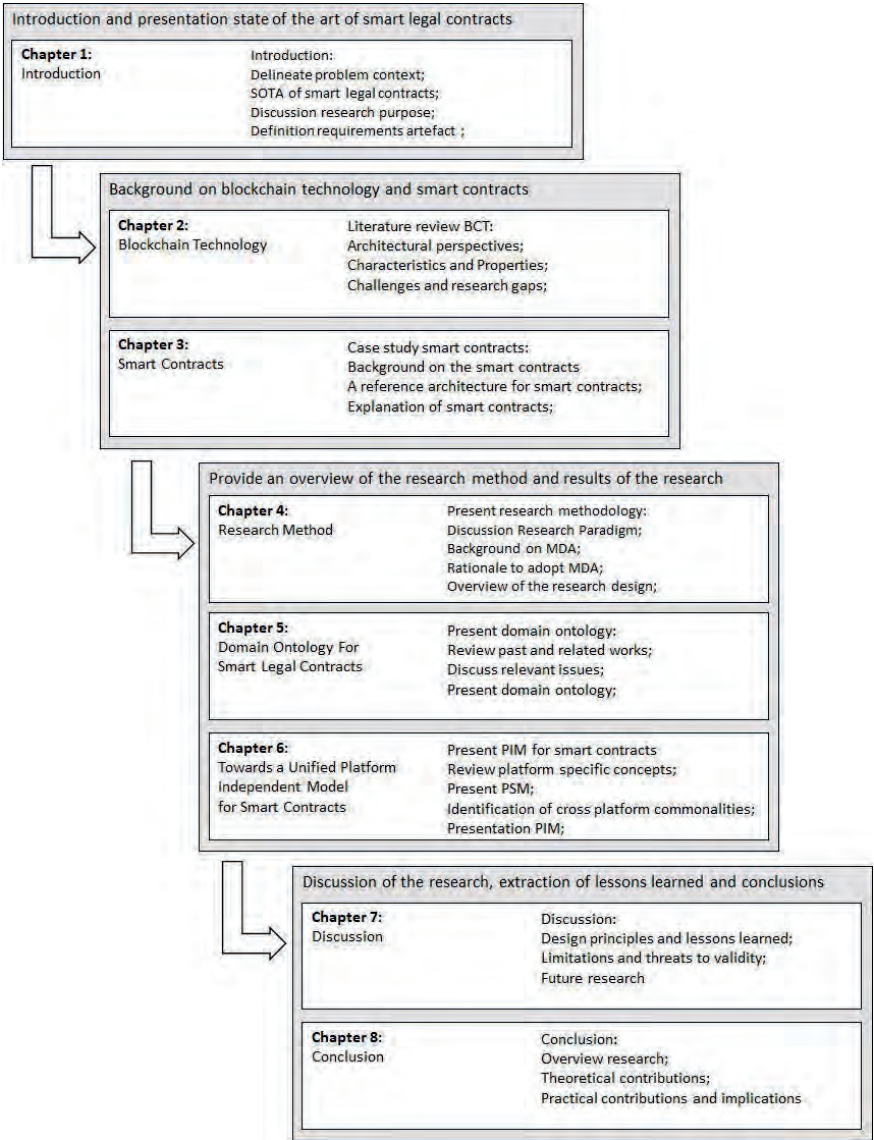


Figure 1.3: Outline and structure of the dissertation

Chapter 2

Blockchain Technology

2.1 Introduction

In 2008 Satoshi Nakamoto, a person or group of people¹, introduced the concept of a peer-to-peer (P2P) version of electronic cash that allows for online payments to be made directly from one party to another without any trusted financial institution [188]. The Nakamoto article preludes the rise of Bitcoin, ushering in the dawn of blockchain technology (BCT). BCT utilizes the concept of a digital distributed ledger (i.e., a record or "book" of transactions), which enables the participants of a P2P network to record transactions that are publicly verifiable. In essence, BCT ensures trust between parties without any trusted intermediary when performing transactions [265]. BCT has gained considerable scholarly attention [281] — for this very reason, we seize the opportunity to conduct and report a systematic and *multivocal* study of the state of the art in BCT for the benefit of further well-founded research as well as practice. As said, we operate a multivocal systematic study, namely, we not only focus on research literature but consider the so-called *grey* literature (i.e., books, technical reports, whitepapers, and more) which may carry important information concerning the BCT software architecture landscape [208].

We flesh out the results of our study starting from a rigorous definition of what BCT is and is not. Indeed, the terms BCT and distributed ledger technology (DLT) are frequently used interchangeably despite attempts to semantically discern them on their distinctive underlying architectures [112]. Second, using the well-known 4+1 software architecture framework by Kruchten [141] as a lens for analysis, the study outlines the design options available in the state of the art for BCT architectures.

Third, through the scenario perspective several applications are presented to illustrate how BCT could be harnessed for different scenarios. By categorizing and summarizing the current applications of BCT practitioners gain more insight in the rich palette of possibilities BCT has to offer. Fourth, with this study we highlight the properties of BCT and elucidate their arising trade-offs. Insights on these properties and their trade-offs aids practitioners in making design choices while providing scholars means to assess blockchain architecture.

Fifth, the research delineates a comprehensive and data-driven overview of the challenges in the field of BCT. Following, this study thoroughly discusses the relation amongst BCT concepts based on a rigorous and systematic analysis of the literature. Establishing this relations can help practitioners further develop BCT while scholars are provided with more accurate measurements to gauge its benefits. The discussion is further strengthened by presenting highlights and observations in-depth attained from the papers under review. Finally, this

This Chapter is based on a peer-reviewed publication in: Butijn, B. J., Tamburri, D. A., & Heuvel, W. J. V. D. (2020). Blockchains: a systematic multivocal literature review. *ACM Computing Surveys (CSUR)*, 53(3), 1-37.

¹To this day the identity or identities of Satoshi Nakamoto remains unknown [278].

research presents a systematic overview of all current research gaps to help direct future research endeavors.

The remainder of this chapter is organized as follows: The next section reviews the background and basic notions of BCT. The methodology section (Sec. 2.3) elaborates on the approach taken to attain the results of this study. In the section thereafter (Sec. 2.4) a definition of BCT that has been constructed based on the literature is presented. Following, the study presents BCT software architecture from multiple perspectives in 2.5. Section 2.6 reviews the scenarios for using BCT found in the literature to provide insight into the applications for which BCT is used. The characteristics of BCT and architectural trade-offs are presented in section 2.7. Challenges of BCT are presented in section 2.8, along with an outlook. The results of the research are discussed thereafter (Sec. 2.9). Based on the discussion of the results, in section 2.10 suggestions for future research are presented. In section 2.11 the limitations of this research and potential threats to validity are addressed. Section 2.12 concludes the chapter.

2.1.1 Related Work

Several previous related surveys exist in the state of the art, even though none of them have the scope, breadth, and width we adopt in our research design. We report the most closely related here below and highlight the novelty of our work. Yli-Huumo et al. [281] review and map the extant literature to indicate research gaps. This review however, predominantly features literature related to Bitcoin and corresponding issues. On one hand, the synthesis operated by Yli-Huumo et al. is only loosely systematic and, on the other hand, the field of BCT has been rapidly developing since the publication of their study [281]. By comparison, our work also fleshes out the architecture of other blockchain networks, and provides a more up-to-date and systematic overview of BCT developments.

In [252] a literature review on cryptocurrencies is presented, which is however not focused on BCT architectures. Moreover, in this work we focus not only on the cryptomarket but consider use-cases for blockchain other than cryptocurrency. The same issue recurs with other reviews focused on the literature related to smart contracts, i.e., programs that can be deployed and run on a blockchain [5] and their applications [166]. For example, Bartoletti and Pompianu [22] focus on smart contract applications, and review the platforms and design patterns for such smart contracts. In our own work, we build from these foundations and include a comparison of smart contracts with the architecture principles of BCT.

Furthermore, there are several literature reviews that examine the use-cases and specific applications of BCT, e.g., Karafiloski et al. [126]. Several other sector- and domain-specific reviews also exist, e.g., for the Internet of Things (IoT) [53, 129], or Multi-Agent Systems (MAS) [46]. While other papers review how BCT could be utilized for the aforementioned domains, these reviews do not present BCT applications based on a rigorous scientific evaluation — in our work we set out to operate a systematic synthesis of architecture elements as well as the alternatives in architecture decision-making spanning multiple domains and encompassing reference literature from, e.g., Supply-chain management [134, 263], usage of BCT by governments [26], BCT in healthcare [145], and more.

By contrast, the work of Tama et al. [246] presents a brief critical review of BCT and some applications for multiple fields. Our work however, provides a more elaborate overview of BCT applications for these fields, and in addition an in-depth insight into the architecture of blockchain technology that has been obtained through a grounded theory approach. Furthermore, our study includes a definition of BCT based on formal concept analysis. What is more, Casino et al. [47] present a review of BCT as a basis for multi-purpose applications

design. Rather than concentrating solely on BCT applications, our work in addition provides a rigorous definition of blockchain based on formal concept analysis and offers an extensive multi-vocal catalogue and accompanying descriptions of anything that was published about blockchain technology. The catalogue presented in this paper follows the well-known 4+1 view framework [141] for architecture description to aid anyone in framing, operating, deciding upon or describing blockchain architectures in general. Hence, the scope of our work is much broader, not only discussing applications of blockchain but in addition its architecture in depth. Finally, we provide a data-driven, in-depth, and evidence based overview of research gaps in the field of BCT.

2.2 Background and Basic Notions

As previously stated, BCT first appeared in 2008, featured in the seminal paper "Bitcoin: A Peer-to-Peer Electronic Cash System" by Nakamoto [188]. The paper proposes a P2P electronic cash system that allowed for the execution of transactions between one party and another without requiring a trusted third party to act as a safeguard and check the validity of the transaction. A year later in 2009, the Bitcoin network was launched [287].

2.2.1 Historical Setting

The first solution that Nakamoto suggested to enable the transactions of digital coins is that owners of a coin wishing to commit a transaction should digitally sign a hash of the previous transaction and the public key of the next owner, both is added to the end of the coin. An electronic coin as such is defined as a chain of digital signatures. By verifying the signatures of a coin the payee can verify the historical chain of ownership. However, this provides a payee with no guarantee that the coin has not already been double spent as there is no way to verify that the previous owners did not sign any earlier transactions. *Double-spending* refers to spending the same currency in two distinct transactions at the same time. In traditional settings, a centralized trusted third party (e.g. bank or mint) verifies whether the owner of a coin did not double spend the same coin. To verify transactions traditional trusted third parties maintain a centralized ledger which records all transactions and the order in which they were enacted. Moreover, the trusted third party needs to be aware of all transactions as there is no other way to confirm the absence of a transaction.

2.2.2 Terms and Definitions

In order for transactions to be executed without a trusted third-party there also needs to be full awareness, and a single history of these transactions. In the Bitcoin paper two solutions are proposed to accomplish the aforementioned goals: (1) Transactions should be publicly announced to all participants in the network. These objectives are attained by employing a distributed ledger on a P2P network. Specific network participants called *nodes* each store a local copy of the ledger. (2) Nodes need to reach a consensus about the history of the transactions, and the order in which they were received. This raises another problem however: Some of the nodes in the network might behave maliciously and try to change the communication contents. In literature this problem is referred to as the *Byzantine Generals Problem* [153]. Non-malicious nodes need to be able to distinguish the information that has been tampered with from the correct information by reaching a consensus over the consistency of the distributed ledger to determine the validity of a transaction. Consequently, this requires proof that when the transaction was executed, the majority of nodes have reached a trustworthy consensus that it was the first received. In essence, these requirements are

introduced to ensure that the system to a certain extent can tolerate malicious behavior by nodes participating in the network, to which is commonly referred to as *Byzantine fault tolerance* [205].

In the seminal Bitcoin paper [188] several concepts are presented to satisfy these requirements starting with the use of a timestamp server. The server takes a hash of the block of transactions that are required to be timestamped and publishes the hash on the network. What the *timestamp* proves is that the data existed at a certain point in time. The hash is *chained* to the previous hash because the latter time stamp is included in the former. As a result, each additional hash reinforces the ones before it, and as more blocks are added the chain will grow ever stronger. The next concept presented enables nodes to reach a consensus on whether the distributed ledgers are consistent with one another, thus that all transactions are valid. A naive way of accomplishing this would be to let the majority of nodes vote over its consistency. However, that would make the blockchain prone to *Sybil attacks* whereby a malicious attacker creates or copies multiple identities in order to control the network.

Bitcoin diminishes the possibility of a Sybil attack by employing a Proof-of-Work (POW) *consensus protocol* which stipulates that not the majority of IP-addresses count as the majority vote of the network, but rather the majority of computational power. While it might be easy for an attacker to create several nodes in a network, amassing large amounts of computational power might prove to be more difficult. The PoW consensus algorithm distributes accounting rights and rewards through a computing power competition in which all nodes of the network can participate. Nodes try to be the first to solve a computational hard mathematical puzzle by finding the right *nonce* (a random number) for the block-header based on information of the prior block. This process is called *mining* and the nodes executing the calculations are referred to as *miners* in the Bitcoin nomenclature. The first miner to finish creates the next block and is rewarded by receiving an amount of Bitcoin. However, because the mining process is probabilistic two or more blocks might be created and propagated by distinct miners simultaneously. These phenomena are known as *forks*. In the event of a fork, nodes as a rule always trust the *longest chain* of blocks as the chain holding the truth with regard to transaction validity (which is analogue to the most computational work). Other nodes wait until new blocks are proposed after the occurrence of the fork to determine which chain will become the longest chain. Consequently, transactions are not confirmed before a longest chain has formed.

The *longest chain rule* is a safeguard to secure the blockchain against the possibility to delay the propagation of transactions which in turn, opens the possibility of introducing fake transactions. As the computational power and interests of the miners might vary the PoW consensus protocol increases or decreases the difficulty of the mathematical problem in such a way that the interval between the generation of new blocks, referred to as *block interval time* remains constant at 10 minutes. Tampering with the transactions recorded on the Bitcoin blockchain would therefore require an attacker not only to be the first one to generate the latest block, but also to control the longest chain.

After the introduction of the genesis PoW-based consensus protocol for Bitcoin many others have been introduced for blockchain such as: (1) Proof-of-Stake (PoS), which replaces PoW based mining with a mechanism which makes the chances of mining a block proportional to the amount of stake (currency) a miner has [88, 157, 244, 282]; (2) Delegated-Proof-of-Stake (DPoS), where the chances of mining a block are also based on a miner stake but allows for the delegation of voting on the correctness of a block [183, 285]; (3) Proof-of-Elapsed-Time (PoET) which used dedicated hardware to create consensus [177, 278], and (4) Zero Knowledge Proofs (ZKP) that aim to provide users performing transactions with more privacy

[112, 274, 282]. The advent of blockchain also revived the interest in preexisting consensus protocols such as Practical Byzantine Fault Tolerance (PBFT), which could be utilized for a similar purpose as ZKPs [72, 274, 285].

Furthermore, Bitcoin was envisioned as a *public blockchain network* that anyone willing can access, and that is *permissionless*, meaning that everyone connected to the network can request transactions or become a miner to check the validity of transactions. By contrast, in the past decade *private blockchain networks* have been introduced that allow only selected participants from one organization to join the network which can also perform only actions that are *permissioned* on the network. Finally, *Consortium blockchain networks* can be considered a hybrid approach as the number of participants that can join the network is restricted, but they can be from different organizations. Among the connected participants the permissions they are granted on the network might differ [211, 274, 285].

Although the Bitcoin technology introduced the concept of BCT to allow for electronic payments using *cryptocurrency* (digital coins) between anonymous peers, nowadays other blockchain networks such as Ethereum offer to possibility to deploy *smart contracts*, that is, programs that can be deployed, run, and verified correct over a blockchain. Smart contracts use triggers, conditions and business logic to enable more complex programmable transactions [274] for the automation of (business) processes [83, 88, 231].

2.3 Research Methodology

To attain our results, we conducted a systematic Multivocal Literature Review (MLR) on blockchain technology. Specifically, we address the following research questions:

- RQ1 How can blockchain technology be systematically defined?
- RQ2 What applications of blockchain technology have currently been published and how can these applications be classified?
- RQ3 What are the properties of blockchain technology and what are their trade-offs?
- RQ4 What are the challenges for blockchain technology?
- RQ5 What are the current research gaps in the field of blockchain technology?

The first research question rotates around providing a systematically-derived definition for blockchains while RQ2 focuses on the applications for which BCT is utilized. RQ3 seeks to offer an overview of the notable properties of BCT (architectural or otherwise) as well as their trade-offs. The fourth research question focuses on delineating the challenges in the field of BCT. Finally, RQ5 aims at presenting research gaps that future research endeavors can address.

2.3.1 Data Preparation Approach

The benefit of a MLR approach is that, beyond typical systematic literature reviews [131] (SLRs) which use academic peer-reviewed articles alone, a MLR also allows for the inclusion of Grey Literature (GL). GL is typically produced by practitioners, such as private industry, governments, academics and industry, and any party which is not controlled by commercial publishers or peer-review. Generally, therefore, grey literature is not published in books or scientific journals. However, this literature can provide invaluable insights into the state of the practice in a field [86]. Given that at the moment of writing the field of BCT is still relatively in its infancy, we therefore deem it appropriate to include relevant literature created by practitioners in the field of BCT for a better understanding of the field. Including GL in our

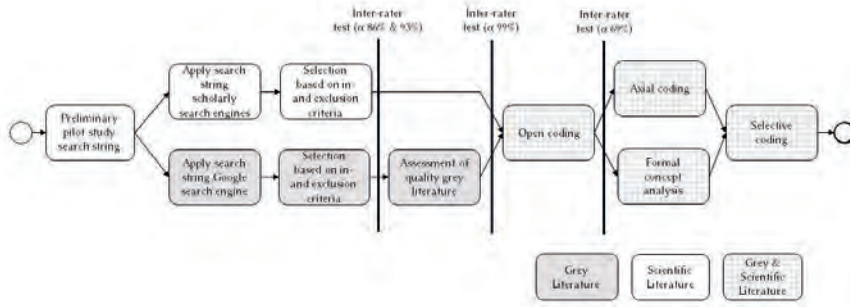


Figure 2.1: Research methodology, an outline.

review allows us to combine and synthesize academic literature with the state-of-the-art in practice.

In conducting our MLR we set out to identify (a) all relevant academic peer-reviewed articles (scientific literature), (b) all relevant grey literature for this study. To reduce the possibility of researcher bias, a predefined protocol for the identification of both the relevant scientific literature (SL) and GL needs to be established [86]. While carrying out our systematic literature review we followed three steps: (1) Create a selection of articles to review. (2) Conduct the review (3) Analyze the data. A process model of the methodology used for this research is depicted in Figure 2.1.

2.3.2 Search strategy

The first step has been carried out using the protocol for systematic reviews suggested by Kitchenham [131]. The protocol suggests three stages for a literature review: (1) elaborate the search string; (2) apply the string on chosen search engines; (3) Filter out and extract primary papers based on pre-established exclusion criteria from search results. The implementation of these steps is presented in Figure 2.2.

The search string was determined by deriving relevant keywords from the research questions. Before carrying out our systematic search, we conducted a preliminary pilot study by experimenting with the search terms to select more results. This process yielded the following search terms:

(1) "Blockchain" ∨ "Blockchains" ∨ "Distributed" ∨ "Decentralized" (2) "Ledger" ∨ "Technology" ∨ "Database" (3) "Applications" ∨ "Use Case" ∨ "Implementation" ∨ "Example" ∨ "Case Study" (4) "Architectural" ∨ "Architecture" ∨ "Form" ∨ "Fabric" ∨ "Structure" ∨ "System" ∨ "Design" (5) "Choices" ∨ "Options" ∨ "Decisions". When combined, the preceding terms were used in the following search string:

$$[(1 \wedge 2) \wedge 3 \vee (4 \wedge 5)] \quad (2.1)$$

In the next stage (2), the search string has been applied to the following scholarly search engines: ACM Digital Library, SCOPUS, IEEE Xplore Digital Library, Science Direct, Springer-Link and Wiley InterScience, EBSCO electronic library, JSTOR knowledge storage and, ProQuestABI/Inform throughout March in 2018. The final stage (3) of the systematic review, the initial results were screened against inclusion and exclusion criteria that are shown in Fig.

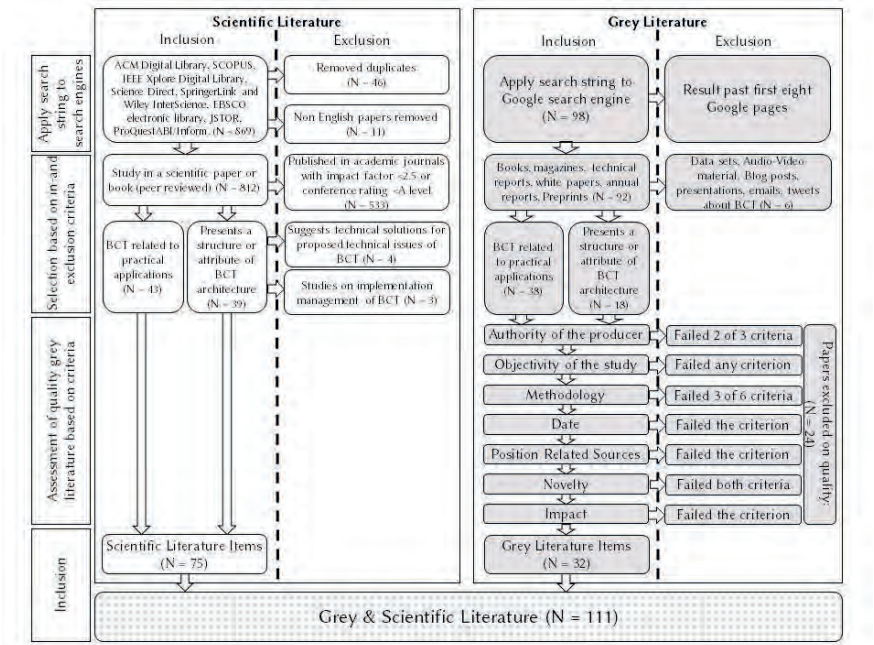


Figure 2.2: Sample Search and selection strategy, a process model.

2.2 (Selection based on in-and-exclusion criteria)². For brevity's sake we have not included a more elaborate description and rationale behind these criteria in the paper, but they can be accessed online (see this link for more details).

For the second part of the MLR, to identify all relevant Grey literature, we established another protocol to filter and extract the GL using the guidelines suggested by Garousi et al. [86]. The protocol has been conducted in three stages: (1) Search process, (2) Source selection, (3) Study quality assessment. The implementation of these steps can be found on the right-and side of Fig. 2.2.

In the first stage, we applied the search string to the Google search engine. The search process yielded 8.330.000 results when applying the first search string ("Blockchain" ∨ "Ledger" ∨ "Applications"). Because of the significant amount of results we initially limited our review to the first eight pages (20 results per page) provided by the Google search engine. Incrementally the next pages thereafter have been reviewed using inclusion and exclusion criteria related to the type of grey literature source (e.g. books, magazines or video files), (see Fig. 2.2). Thereafter the pages were incrementally reviewed by title and abstract, starting from the first results page using the inclusion and exclusion criteria depicted in Fig. 2.2 (selection based on in-and exclusion criteria). If <50% of the results on a page were not relevant for this research, the search was stopped there. We further refined the GL studies we obtained from the first eight Google pages using the same inclusion and exclusion criteria.

²The (N = followed by a number) in Fig 2. represents the number of papers included or excluded based on the selection criteria.

Table 2.1: Quality criteria grey literature.

| Category | Exclusion Criteria | Criteria to Satisfy |
|---------------------------|--|---------------------|
| Authority of the producer | The publishing organization is reputable, or the individual author is associated with a reputable organization | 2/3 |
| Objectivity of the study | The author has published other work in the field | 3/3 |
| | The author has expertise in the area (e.g. job title) | |
| | The statement of the sources is objective | |
| Methodology | There are no vested interests | 4/6 |
| | Conclusions are supported by data | |
| | The source has a clearly stated aim | |
| | The source has a clearly stated methodology | |
| | The source is supported by authoritative, documented references | |
| | Limits are clearly stated | |
| Date | The work covers a specific question | 1/1 |
| | The work refers to a particular population | |
| | The item has a clearly stated date | |
| Position related sources | Key related GL or formal sources have been linked/discussed | 1/1 |
| Novelty | The item enriches or adds something unique to the research | 1/2 |
| | The item strengthens or refutes a current position | |
| Impact | The GL source should have citations and backlinks to substantiate the arguments made in the study | 1/1 |

During the second phase, we assessed the quality and relevance of the sources of the primary GL we obtained since it cannot be assumed that the quality of GL is guaranteed. Exclusion criteria suggested by Garousi et al. [86] have been used for this purpose (see Fig. 2.2).

The exclusion criteria used consist of 7 quality categories ranging from the authority of the producer to the objectivity of the study that can be found in Fig. 2.2 under assessment of quality grey literature. Combined these quality categories encompass 17 criteria that have been assessed one by one for each GL item. An overview of all the quality categories, quality criteria, and how many of these criteria had to be satisfied to include the item can be found in Table 2.1.

The authors of this study (viz., the first two authors of this study), have indicated whether a GL item: (a) satisfied, (b) did not satisfy a criterion. In the cases where one of the criteria could not be assessed (e.g. because this information was missing) we have assessed these criteria as if they did not satisfy the criterion. GL items that did not satisfy the threshold for each quality category were excluded from the sample. After the selection process we merged the grey-and scientific literature into one sample as the literature under review.

2.3.3 Data Analysis

This section details the analysis methods enacted to address our research questions.

2.3.3.1 Formal Concept Analysis

To address RQ1, a Formal Concept Analysis (FCA) approach was adopted. FCA is a systematic approach to derive a formal ontology or concept hierarchy from a set of objects and their attributes [175]. A complete description of the FCA method employed to attain our definition of BCT can be found in Appendix A.1.

2.3.3.2 Grounded-Theory Analysis

For step 2 and 3 of the MLR, and to address RQs 2,3 and 4 a Straussian Grounded Theory (GT) approach [93] was adopted. In the scope of straussian GT, a series of systematic steps are enacted to allow a theory to emerge from the data (hence, "grounded") using codes. For our research each code represents a concept or theme related to BCT. Whenever a paragraph in the literature under review represented one of these concepts or themes, the related appropriate code has been attached. In GT this process known as "coding", and includes the phases that are described in Appendix A.2.

2.3.4 Inter-Rater Reliability Assessment

We employed Krippendorff coefficient (or $K-\alpha$) [108], to evaluate the inter-rater reliability of the inclusion and exclusion of SL items, the in- and exclusion of GL items, the quality assessment of the GL, and the coding process of the pilot study. The coefficient measures the agreement between two ordered lists of codes which have been applied as part of content analysis. The methods used to assess the inter-rater reliability between the raters, and the results thereof can be found in Appendix A.3.

2.3.5 Sample Selection Results

This section outlines the sample results of our search strategy (Sec. 2.3.2). First the section presents the distribution of the sample between grey and scientific literature, along with the distribution the publication venues per year. Thereafter, the section showcases a frequency analysis of the topics discussed in the papers under review along with an overview of these trends per year. Finally, the distribution of these topics between grey and scientific items is presented and discussed.

2.3.5.1 Publication Venues and Distribution Literature

After the assessment of the GL, the search strategy yielded a total of 33 GL items and 78 of scientific peer-reviewed papers selected for this study. From this point on, we no longer distinguish the results whether they were derived from GL or SL but flesh out results over the total of 111 studies be the object of this research. Figure 2.3 depicts the number of SL and GL per publication year. The results depicted for the year 2018 have only been collected until March 2018 and therefore might skew the results.

Figure 2.3 shows that from 2008 until 2013 BCT has gained little attention from either practitioners or scholars. The figure show an overall increase in the SL and GL published from 2014 onwards. Furthermore, the statistics show that there is a growing interest from the scientific community for research in BCT. More specifically, from 2017 onwards twice as many articles have been published as compared to the years before. However, these results also indicate that research in the field of BCT is still in its infancy given that from 2008 to 2016 little scientific work has been published. The search and selection results also indicate an increase in the amount of practitioners literature being published. Furthermore, the sources from which these research items were identified for this study are diverse (see Fig. 2.4), ranging from articles in technical magazines, books, and technical reports alike. The majority of items however, were published in conference proceedings and reflect white literature.

In the years directly following 2008, i.e., the introduction of BCT, publications on the topic were almost evenly distributed among different sources (e.g. books, conference papers). However, as of 2016, books on BCT have not been found by this research. Although the

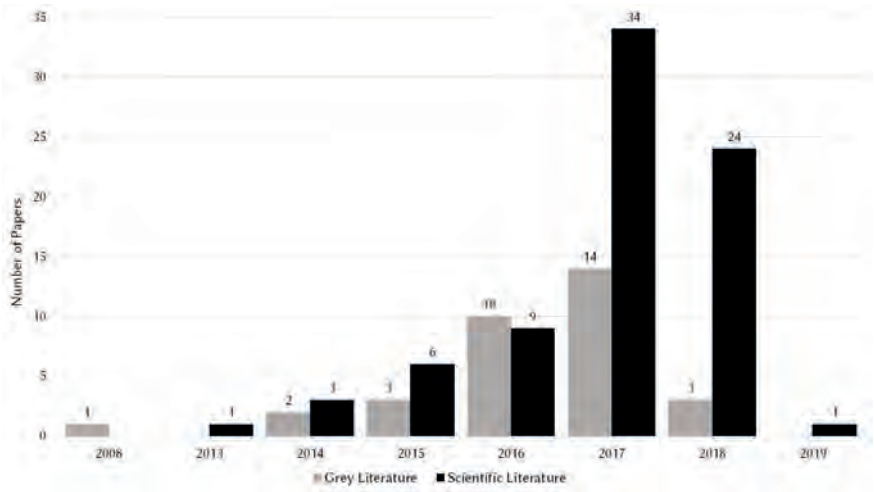


Figure 2.3: Sample results; Grey and Scientific Literature across primary studies.

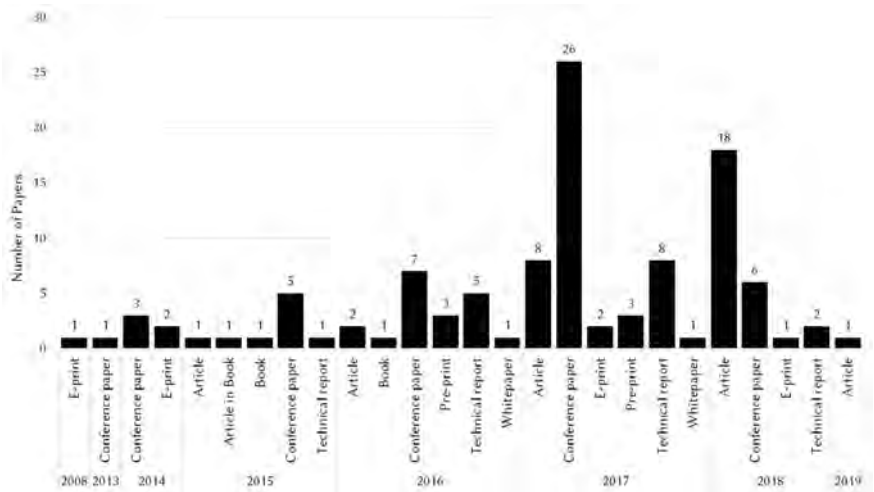


Figure 2.4: Sample results; Publication Venues per Year.

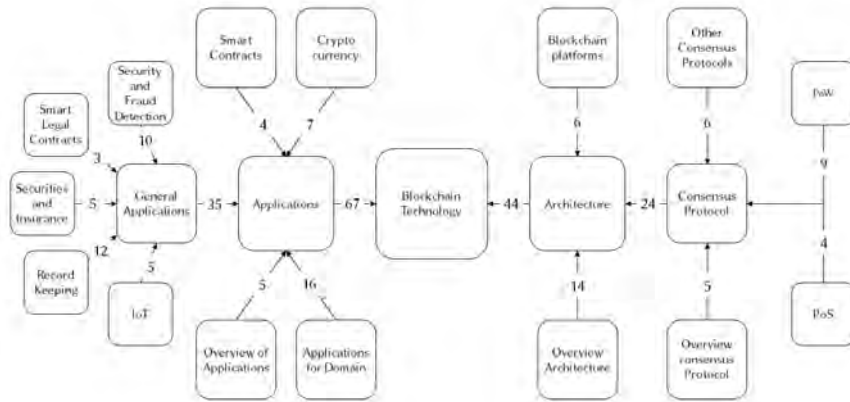


Figure 2.5: Sample results; Topics frequency analysis.

publication of conference papers shows an increase in 2017, articles published in scientific venues are more gradually increasing in frequency.

2.3.5.2 Topics in the Field of BCT

Fig. 2.5 shows the the main topics of the papers under review, as elicited using a grounded-theory approach. More specifically, the blocks in Fig. 2.5 represent the topics found in the literature, while the number on the arrows between the blocks represents the *weight* of the topic, in terms of number of papers where those topics were coded. The direction of the arrow itself depicts under which of the composed main topics the sub topics are categorized. Among the items selected for this study, BCT-based applications are strongly represented (see left-hand side of Fig. 2.5) while items on BCT architecture and are represented slightly less, with a ratio of 2/3.

In terms of applications, five sub-themes can be distilled: (1) cryptocurrencies, (2) smart contracts, (3) papers that provide an overview of BCT, (4) literature that suggests applications of BCT for specific domains, and (5) finally, literature that presents general applications for BCT. Interestingly, Fig. 2.5 again shows that the applications of BCT for a specific domain and general applications of BCT has gained considerable attention. On the one hand, literature on BCT for specific domains encompasses complex domains such as e-government, the financial sector, and relief development [52], while literature on the general topic of smart contracts is rather limited. On the other hand, the specific technical architecture literature over BCT reflects five categories: (1) security and fraud detection; (2) smart legal contracts; (3) securities and insurance; (4) record-keeping; (5) the Internet of Things (IoT). Among these categories the majority of papers has been published on utilizing BCT for record-keeping, that is, registering certain data on the blockchain to ensure its immutability. Another major focus of research on BCT applications is security and fraud detection, these items focus on using blockchain for safe distribution of data among peers. A more detailed description of the contents of literature of applications for BCT can be found in section 2.6.

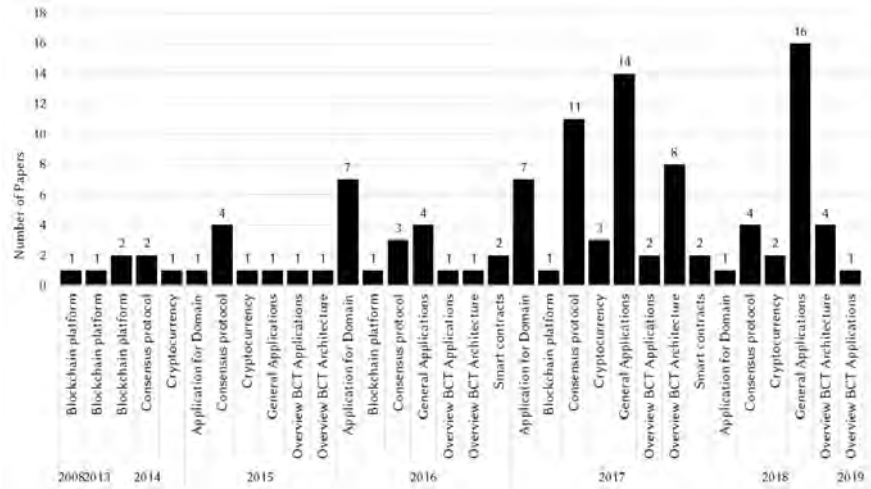


Figure 2.6: Trends in BCT publication topics.

2.3.5.3 Trends in Publications on BCT

Four key trends in the literature under review can be observed (see Fig. 2.6). First, there is a balance in the distribution among topics even though the overall number of works on BCT is steadily increasing.

Second, two exceptions are (a) works related to applications for specific domains and (b) general applications research, as previously discussed — for these, the years 2016 to 2018 have seen a tremendous increase in research and practical work.

Third, publications on blockchain-based smart contracts have seen a rise only from 2016 onwards. A similar observation can be made for items related to cryptocurrency as only one paper was published in 2015, with 3 papers being published in 2018.

Lastly, Since 2014 an increasing amount of papers on the topic of consensus protocols have been published, a trend that continues to date.

What is more, in terms of past publications by both practitioners and scholars have predominantly been focused on applications for BCT (see Fig. 2.7) with more of the grey literature being published on applications (72%) as compared to scientific literature (55%).

With respect to engineering research focus, scholars have focused their efforts on consensus protocols whereas practitioners have mostly presented works on blockchain platforms or overviews of BCT architecture.

2.4 A Systematic Definition of Blockchain Technology

So far we have referred to BCT as a single technology, however BCT is a clever combination of several technologies and elements and there is no consensus on the definition of a blockchain [112], with a precise definition of blockchain technology often subject to controversial and subjective opinion [243]. Stemming from the data available to us, We strived to construct a rigorous definition of BCT based on (a) the identified software elements drawn

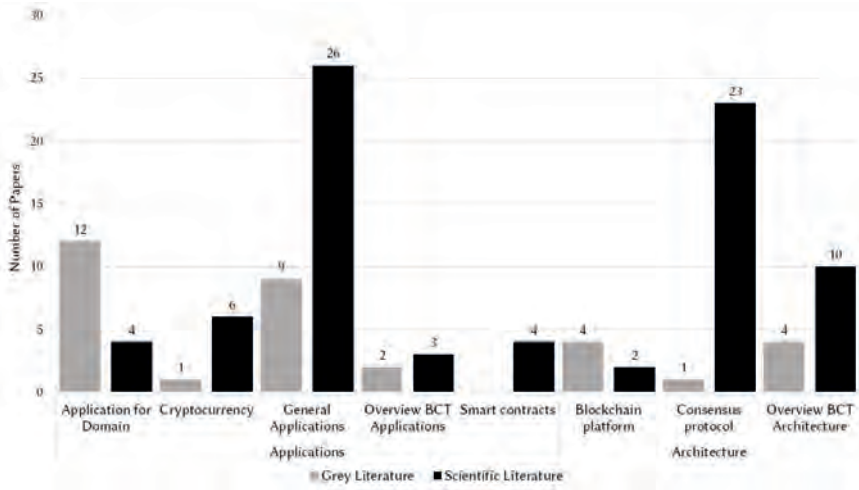


Figure 2.7: Distribution topics scientific and grey literature.

from literature, (b) relations among them and (c) their properties³ reflecting the software architecture research and practice state of the art [24].

First, our data indicates that: *Blockchain technology* is a form of *distributed ledger technology*, deployed on a *peer-to-peer network* where all data is replicated, shared, and synchronously spread across multiple peers. The technology allows actors participating in the network to perform, sign, and announce *transactions* by employing *public key cryptography*. Transactions are executed following a *consensus protocol* operated by specific nodes to ensure the validity of transactions requested by other peers in the network, and to synchronize all shared copies of the distributed ledger. During a consensus protocol execution, the data of valid transactions, along with other required metadata concerning the network, and the hash of the previous block is bundled into a *block* using *hashing functions*. The essential and key property reflecting BCT architectures is that each block contains the hash of their predecessor, therefore linking all prior transactions to newly appended transactions; the blocks therefore form a *chain* with the aim of establishing a tamper-proof historical record.

2.5 Blockchain Technology: Architecture Elements

This study examines the blockchain architecture landscape, arranging the elements found in literature through the well-known 4+1 software architecture framework introduced by Kruchten [141]. The Kruchten framework delineates the comprehensive interplay of relations, properties and software elements in BCT and encompasses five views, namely: (1) logical view; (2) development view; (3) process view; (4) physical view; (5) a use case view. In Appendix B.1 a more elaborate description of these views is provided.

Using the logical view, we first delineate the architectural elements to present the functionalities that various end users ultimately use from a blockchain. Further on, the development view describes how building BCT can be divided into smaller chunks of programmable code.

³Public-key cryptography is the commonly used to describe the exchange of information using a set of private and public keys. Hence, when constructing a definition the term public-key cryptography has been used instead of private and public keys as an attribute of BCT.

Subsequently, the process view shows how IT systems behave during run time and is of interest to system integrators that need to know about the thread of control to execute operations utilizing BCT. Beyond that, the physical view is of interest to system engineers that maintain overall blockchain system, also, given that BCT completely resides on a P2P network its different arrangements is discussed in the physical view. The use-case view of the 4+1 model is recapped later in Sec. 2.6.

2.5.1 Logical View

The logical view emphasizes on the functional requirements and services the system should provide to its end users [141]. Decomposition of the architecture aids in identifying the elements that are common across the system. We used an ontology for BCT as proposed in [94, 143] to organize the discussion of its main elements. A more elaborate, and in-depth description of BCT elements is provided online (see Appendix B.1 how to access the material).

Blockchains are transaction oriented. Transactions in a blockchain system are executed using *public key cryptography*. *Cryptographic Hash functions* are used for the purpose of many operations, such as signing transactions (SHA-256 in the Bitcoin case[188]). The peers in the P2P network, also referred to as nodes are devices capable of processing and verifying transactions. Depending on the permissions all nodes or a specific subset of nodes validate transactions. The *permissions*. There exists at least three categories of blockchain networks [274, 285]; Public, private, and consortium networks that have different arrangements in terms of their permissions. On a blockchain transactions are stored in *blocks*. Each block is linked to its predecessor known as *parent block* by including its blockheader hash to form an integral chain of blocks that can be traced back to the first, or *genesis block*. Hence the term "blockchain" technology. Novel blocks are generated using a consensus protocol. Provided that the transactions included in the newly proposed block are valid, each new block enhances the security guarantees of the block before it [183, 188, 189]. Updates and changes to the software of a blockchain are called *forks*.

2.5.2 Development View

Existing blockchain networks can be leveraged to build Decentralized Applications (DApps) upon that use their services. Developers seeking to build their own blockchain platform have to program multiple software packages. Enabling transactions forms the basis for any blockchain network. A wallet needs to be programmed to allow clients of the platform to interact with other peers in the network. An *address propagation* method should be installed for nodes to interact. Next the nodes need to connect via *peer discovery*. Another aspect is the mechanism for *propagating transactions* [33].

Data with regard to transactions can be stored in two ways: As a first method, like the Bitcoin, one can choose to add data into transactions. Another second method is to add data into contract storage like Ethereum [277]. Finally, an existing consensus protocol can be selected to process transactions or the protocol can be designed from scratch. Online material that can be found via a link in Appendix B.1 further describes the development view.

2.5.3 Process View

The process view specifies which thread of control execute the operations of the classes identified in the logical view. The consensus protocol is at the heart of all BCT processes since it allows for the enactment of transactions and ensures that the distributed ledger

remains consistent. Online materials to whom a link is provided in Appendix B.1 delineates the steps, issues and potential variants of the consensus protocols discussed in this section more in detail.

2.5.3.1 Practical Byzantine Fault Tolerance

(PBFT). PBFT is mostly used in a private setting for permissioned blockchains because it assumes authenticated nodes [72, 274, 285]. The protocol itself is exclusively based on communication, and nodes go engage in multiple rounds of communication to reach consensus [72]. Nodes do not get a reward for achieving consensus, rather in the event of malicious behavior by an authenticated node it can be held legally accountable [112, 209]. A primary leader node mines the blocks. The leader can be changed by other nodes via a "view-change" voting protocol, in the occurrence of a crash or when it exhibits malicious behavior [53, 183].

2.5.3.2 Proof-of-Work

(PoW) is often referred to as the *Nakamoto consensus protocol* [121, 163, 164, 204]. The PoW consensus protocol is designed for the case where there is little to no trust amongst users of the system [278]. Public blockchains need to have a high degree of Byzantine fault tolerance as users can not trust one another.

Consensus in PoW is achieved through a hashing competition between miners. Competing miners need to commit computing power to calculate the solution to the same mathematical problem. To incentivize miners to participate in the consensus process the miner that is the first to find the solution to the mathematical problem reserves the right to publish the next block, and is rewarded by an amount of cryptocurrency [36, 185, 274, 278]. In addition, the miner to win the competition with its peers is also be able to collect the transactions fees that were paid by clients.

Finding the solution to a PoW problem is a computationally arduous process for which there are no shortcuts [189, 278]. The solution to the problem is hard to find, yet easy to check once they have been found [163]. Given that only one miner can win the competition and is rewarded the other nodes have simply wasted resources (CPU power and energy) in their attempt [88, 183, 244, 274, 278]. In addition, because the difficulty of PoW problems increases over time makes it even harder to win the competition [278].

2.5.3.3 Proof-of-Elapsed Time

(PoET) PoET is designed to address the inefficiency of PoW and replaces it with a protocol that is based on *trusted hardware*. A node that uses trusted hardware however, can be checked for certain properties such as whether it is running a certain software. This aids in relaxing the trust model in settings where the Byzantine's Generals Problem might be present [72]. Sawtooth Lake, a project by Hyperledger, leverages Intel's *Software Guard Extensions* (SGX) to establish a validation lottery that makes use of their CPUs capability to render a timestamp that is cryptographically signed by the hardware [115].

2.5.3.4 Proof-of-Stake

(PoS) As a response to the limitations of PoW the BCT community has turned towards Proof-of-Stake (PoS). The PoS consensus protocol has been introduced for public settings [183] with the aim to safeguard against Sybil attacks and malicious behavior by untrusted nodes [72]. The PoS protocol offers a more efficient and environmental friendly alternative to PoW

as computing power is partially substituted by virtual resources (e.g. cryptocurrencies) that miners must invest to propose blocks [88, 157, 244, 282]. Rather than using computer power as a scarce resource to generate security, Proof of Stake uses the scarcity of the coin itself. Therefore nodes that participate in a PoS consensus protocol are more commonly referred to as *forgers* instead of miners [21, 158].

The idea behind the PoS model is that the more assets (e.g. cryptocurrency), or *stake* a node has, its incentive to undermine the system diminishes because subverting the system would inherently mean that the worth of the nodes' stake would decrease [177, 278]. Logically, this implies that one cannot participate in the consensus protocol without owning a stake [88]. A shared commonality of all PoS variants is that nodes that have more stake have a higher chance of generating new blocks [157, 183, 278, 285]. In other words, the more skin a forger puts in the game the higher its reward will be.

2.5.3.5 Delegated-Proof-of-Stake

(DPoS) Delegated Proof-of-Stake introduces another variant of PoS [183, 285]. In DPoS stakeholders elect delegates, referred to as *witnesses* to forge and validate blocks in round-robin fashion [158].

Compared to PoW and Pos, DPoS is more energy efficient. Further, because the voting about the validity of a block is delegated and fewer nodes are needed to validate the blocks can be confirmed more quickly. Hence, as compared to PoW and PoS, DPoS has a low latency. Moreover, parameters including block size and block intervals can be adjusted by *committee members* of the governance board. When a delegate acts malicious this dishonest delegate can be voted out by all the other nodes [158, 285].

2.5.3.6 Zero-Knowledge-Proofs

Recently, different Zero-Knowledge-Proofs (ZKP's) based BCT networks have been proposed to preserve users' anonymity and confidentiality of transactions [274]. In general, ZKP's aim to confirm a statement about a transaction such as "This is a valid transaction" without revealing anything about the transfer (statement) itself or the parties involved [112, 274, 282]. Zerocoin was the first initiative with the aim of providing transaction unlinkability using ZKP's [72]. Similar to the Bitcoin Zerocoin uses the PoW consensus protocol to validate transactions. A *cryptographic mixer* is implemented for Zerocoin to conceal the links between a zerocoin and the corresponding Bitcoin.

Building on the ZKP approach as a foundation, Zcash, extent the privacy guarantees, and improve the efficiency (throughput and latency) of Zerocoin. Zcash uses a variant of the PoW called Equihash. Transactions made using Zcash, including the split and merge transactions, are fully private [72]. Zcash employs a technique called *Zero-Knowledge-Succinct Non-Interactive Argument of Knowledge* (zk-SNARKS) to provide these privacy guarantees [183, 282] that are a specific type of ZKP.

2.5.4 Physical View

The physical view is concerned with the topology of software components and their physical connections. Electronic devices known as nodes constitute a blockchains' P2P network and are the only physical connection to the non-digital world. P2P networks on which blockchain platforms are run have different arrangements; First, the network can be categorized on the basis of permissions (authorization). Second, networks can be categorized with

regard to their accessibility. Permissions to perform operations on the blockchain might differ ranging from allowing anyone to read, write and to partake in the consensus protocol to only one of these permissions. Control over these permissions can be confined to a distinct group of nodes, or all nodes.

As the name suggest *Permissionless* grant permission to all nodes in the P2P network to read and write transactions. *Permissioned* blockchain platforms have confined and idiosyncratic permissions for their nodes [181, 231, 274, 278].

The P2P network can also be described from the perspective of network accessibility. In the literature three categories of P2P networks can be distinguished that are coupled to a permission model [44, 112, 143, 173, 274, 285]. A *Public blockchain*, like the Bitcoin or Ethereum have *open network access* meaning that anyone willing is allowed to join the network. *Private blockchains* are blockchains networks that are owned by one organization. Contrary to public blockchains access is confined. *Consortium blockchains* are similar to private blockchains in the sense that nodes first need to be authenticated before granted access to the network. However, consortium blockchains allow nodes from different organizations to access the blockchain network [183, 211, 274]. A more elaborate description of both models and blockchain networks can be found in Appendix B.1.

2.6 Blockchain Use-Case View: Main Usage Scenarios

The use-case view aims at providing a description of an architecture by illustrating an essential set of use cases and scenarios for their usage. Our data suggests there are mainly three flavours of BCT applications: (1) Cryptocurrencies, (2) Smart contracts and (3), general-purpose applications. Following these versions we further categorize BCT applications [244]. For instance, the general-purpose applications of BCT can be arranged into five additional categories that encompass: (1) Security and Fraud Detection, (2) Securities and Insurance, (3) Record-Keeping, (4) Internet-of-Things, (5) Smart Legal Contracts.

2.7 Blockchain Technology: Main Architecture Properties

The scope of our analysis revealed 8 essential architectural properties with a directed mutual influence relation evident from the state of the art. In fact, stemming from the relations our GT analysis we marked with a \rightsquigarrow operator the mutual implication relation evident between the following couples of properties:

- *Decentralization* \rightsquigarrow *Disintermediation*. In traditional centralized transaction systems each transaction needs to be validated by a (trusted) third party (e.g., a bank). The decentralized workings of BCT enables the direct transfers of digital assets between two counter parties without this third party leading to direct disintermediation [88, 189, 231, 285].
- *Programmability* \rightsquigarrow *Automation*. BCT allows for the execution of pre-defined conditions that are automatically executed once certain conditions have been met. BCT enabled smart contracts extend this concept further by allowing (Turing complete) programmability of transactions [58, 115, 231]. The Bitcoin blockchain predominantly offers a service to exchange cryptocurrency and, accordingly provides limited support for smart contracts Blockchains like Ethereum or Kadena offer a fully programmable smart contract environment [112, 177, 189, 231]. Offering smart contracts as a service however, adds another layer of complexity; smart contract execution puts a higher

strain on the data storage requirements, throughput and latency of a blockchain network [61, 112, 143]. Furthermore, arbitrary code leaves room for human errors, and thus increases the chances of bugs [88, 143, 164]. In sum, the degree of automation depends on the services provided which is closely linked to the *design* of a blockchain platform but at the cost of additional complexity.

- *Transparency* \leadsto *Auditability*. Each node in a blockchain P2P network holds a complete copy of the distributed ledger making all transactions transparent [36, 88, 244, 272]. However, for permissioned blockchains permissions to read the ledger can be confined to increase transaction privacy. Decreasing the transparency of the transaction records makes permissioned blockchains less auditable [173, 272, 274]. In short, the auditability of the network depends on the permission *arrangement* of the P2P network.
- *Immutability* \leadsto *Verifiability*. The entire history of transactions performed is recorded and stored in blocks. Given that these blocks are cryptographically chained using hashes, the record becomes immutable [173, 189, 231, 243, 247]. Provided that the entire history of transactions is auditable, the proof that any transaction has (not) taken place in the past is thus verifiable since blockchains are append only [88, 193, 243]. An insecure consensus protocol that allows for the introduction of blocks containing double spend transactions could jeopardize the immutability of the ledger. Therefore the immutability of a blockchains distributed ledger depends on how transactions are *processed* during the consensus protocol [189, 231, 278].

2.8 Blockchain Technology: Challenges and Outlook

Despite being a promising novel technology, currently BCT faces several challenges that inhibit widespread adoption. This section highlights and discusses the challenges evident from the literature.

2.8.1 Latency

One of the challenges BCT faces is that most consensus protocols have a high latency, meaning that the time between the submission of transactions and their confirmation is high [88, 94, 112, 177, 185, 274]. This is due to the fixed blocktime interval for most blockchain networks. Effectively this means that on average it takes the Bitcoin network roughly 60 minutes before transactions are settled and can be regarded as final [112, 274]. Ethereum has made significant process in this area using the Greedy Heaviest Observed Subtree (GHOST) protocol by increasing the block interval to 14 seconds and transaction finality after 12 blocks [88, 274].

What is more, currently the finality for clearing and settling transactions is a legally defined moment. When using BCT to enact transactions settlement finality is probabilistic; The longer a transactions is considered settled by network participants, the less likely it will become that the transaction will be reversed or declared invalid [112, 181]. Clearly, these two arrangements are at odds. A direction that is currently being explored to concurrently address the throughput and latency issues of BCT is that of *sharding* the mining network. ELASTICO [162] is an example of a consensus protocol that shards the mining network. When sharding the network miners are uniformly partitioned into smaller committees that process a specific set of transactions. Accordingly transactions can be processed in parallel and thus throughput capacity can be increased.

2.8.2 Throughput

The maximum throughput of transaction has also been shown to be a challenge [112, 183, 272, 274, 285]. The concurrent throughput challenges for BCT are closely related to those of the latency. At the time of writing the Bitcoin network can reach a throughput of 7 transactions per second [72, 112].

Yet again this problem is related to the blocktime interval but also to *blocksize*. The size of a block determines how many transactions can be included. For the Bitcoin the size limit of a block is 1 MB [274]. Recently there have been proposals to increase the throughput of the Bitcoin blockchain by increasing the blocksize from 1MB to 8 MB [274]. Proponents and opponents of this proposal have interchanged various arguments that so far has reached no conclusive upper-hand [247]. By implementing the *GHOST protocol* Ethereum has managed to improve it's throughput capacity to 15 transactions per second because the block time interval is smaller (14 seconds). Rather than following the longest chain, in GHOST a miner weights the branches in terms of the computational power spend to create them and chooses the better one to follow. Another promising novel development is the introduction of *off-chain payment channels* such as Raiden⁴, Bitcoin Lightning⁵ and Sprites [180] that enables two parties to directly and privately maintain a two-party micro payment channel. Khalil and Gervais [128] extend the concept of off-chain payment channels by suggesting a novel approach that enables the refunding of existing payment channels when they are depleted without performing a transaction on the blockchain network. Recently the segregated witness (SegWit) proposal has been suggested in the Bitcoin community to change the internal design of blocks to increase the throughput of transactions. The proposal entails separating (segregate) signatures (witnesses) from the remainder transaction data. In this manner the size of the witnesses does not add to the data size limit of the blocks [274].

2.8.3 Data Storage

Another challenge that is pointed out by both practitioners and scholars alike is how to cope with the evergrowing need for *data storage space* [88, 112, 154, 244, 287]. This challenge mainly stems from the fact that to verify transactions, a node needs to be aware of the whole blockchains' history. If the Bitcoin were to process an equal number of transactions as Visa the amount of storage required would grow by 214 PB per year [244]. Some suggestions for improvement of data storage have been made such as the introduction of *lightweight clients* that do not download the complete record of transactions. Instead, lightweight clients download only the blockheaders to validate transactions. To verify transactions these nodes use a technique called Simplified Payment Verification (SPV) [278, 285].

2.8.4 Data Privacy

Preserving privacy of participants and confidentially of their data has turned out to be a fundamental challenge [62, 112, 176, 185, 243, 272, 274, 285]. Although transparency is one of the key characteristics of especially public blockchain networks it is at odds with privacy. For public blockchains by design every transaction needs to be visible to every participant for the sake of public verifiability [7, 112, 278], though they can be encrypted and the identity of the user is hidden. In order to address this problem private and consortium blockchains such as Hyperledger [7] and Corda [42] with a permissioned model have been introduced [189]. Another approach solve this problem is the usage of mixers and ZKP (see 2.5.3.6) [20]. In a study [112] 57% of the respondents stated that implementing privacy-enhancing techniques

⁴www.raiden.network

⁵www.lightning.network

in their BCT systems is planned for the future. Out of these respondents 78% have expressed the desire to implement zero-knowledge proofs (ZKP). A second privacy challenge is that a blockchain ledger is *immutable*; Once a transaction has been stored in a block it can not be removed. Further, in permissionless blockchain every node is able to view all transactions and, consequently explore the entire history of transactions. The General Data Protection Regulation (GDPR) however, enforces restrictions on how information about EU citizens may be used and stored [194]. One of the rules that would be difficult to comply with is the "right to be forgotten" that allows an individual to demand the erasure of information under certain conditions. Clearly, the immutability of a blockchains ledger is incongruent with the right to be forgotten [115, 243].

2.8.5 Governance

The governance of a blockchain with regards to *updating* its fundamental rules is problematic [189, 278, 282, 287]. A prime example is the ongoing debate within the Bitcoin community about the block size which has ended in a stalemate [173, 247]. Even for centralized systems updating software can be difficult let alone when a system has many users, geographically dispersed, as can be the case with BCT [278]. Another classic example of the governance problems blockchain currently faces is the response of the Ethereum community to the DAO hack⁶. Due to unintended flaws in the semantics of a contract an attacker was able to obfuscate a large amount of Ether worth an estimated \$50 million⁷. In response to the attack, a hard fork was proposed to recover the Ether, to which 89% of Ether-holding voters gave their consent. Some of the remainder non-consenting voters rejected this fork of the blockchain mostly for philosophical reasons, including the principle that a blockchain is immutable. These voters decided to use the unforked Ethereum blockchain resulting in a split into two separate currencies: Ether (containing the hard fork) and Ethereum Classic (no hard fork) [189, 278]. Another governance issue that needs to be addressed is that of *key management*; BCT is decentralized and as such when a user forgets their private key there is no central authority to recover it [115]. As a solution to this problem He et al. [109] present a wallet-management system based on semi-trusted social networks to recover wallets and the keys they hold. However, the true Achilles heel with regard to private key management is related to the wallets that store these key's; If the hardware on which a users wallet get's lost, targetted with malware or is attacked the private key might get lost or stolen [20].

2.8.6 Usability

A more practical challenge that hampers the widespread adoption of BCT is the current lack of end-user support (BCT is hard to use and to understand) and adequate developer support (few developers tools available) [176, 185]. In line with these observations, research by Tapscott and Tapscott [247] indicates that many Dapps are not accessible to the average person and that interfaces are user-unfriendly. Further, in their study they suggest that there are approximately between 1000 and 2000 developers that understand how to develop Dapps. However, one of their interviewees stated that this number could perhaps increase by establishing creative educational programmes.

⁶The term hack must be qualified; The attacker exploited a vulnerability in the smart contract that allowed a split function (enabling the withdrawal of funds from the contract) to be called repeatedly in order to withdrawal more funds than entitled to. For further reading about the DAO hack the author recommend reading Annex B in [115].

⁷Tapscott and Tapscott [247] argue that the total worth of the obfuscated Ether was around \$70 million, whereas Gatteschi et al. [88] suggest that it was the equivalent of \$60 million.

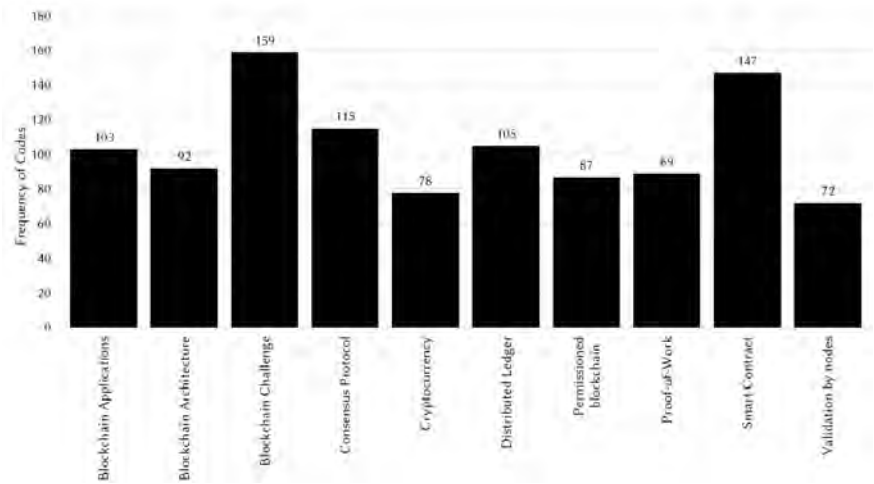


Figure 2.8: Frequency of top 10 most recurring codes.

2.9 Discussion

Our GT-based analysis was used to populate the illustrated 4+1 views, properties and challenges of BCT. First, this section grounds the insights on the 4+1 views of BCT architecture elements, properties, and challenges through discussion. More specifically, the GT-driven 4+1 perspectives on BCT are deepened by discussing coding frequencies and trends of the concepts in literature. Secondly, the section presents observations we made in the scope of our analysis derived from examining the distribution of the topics found in the sample and synthesizing their contents.

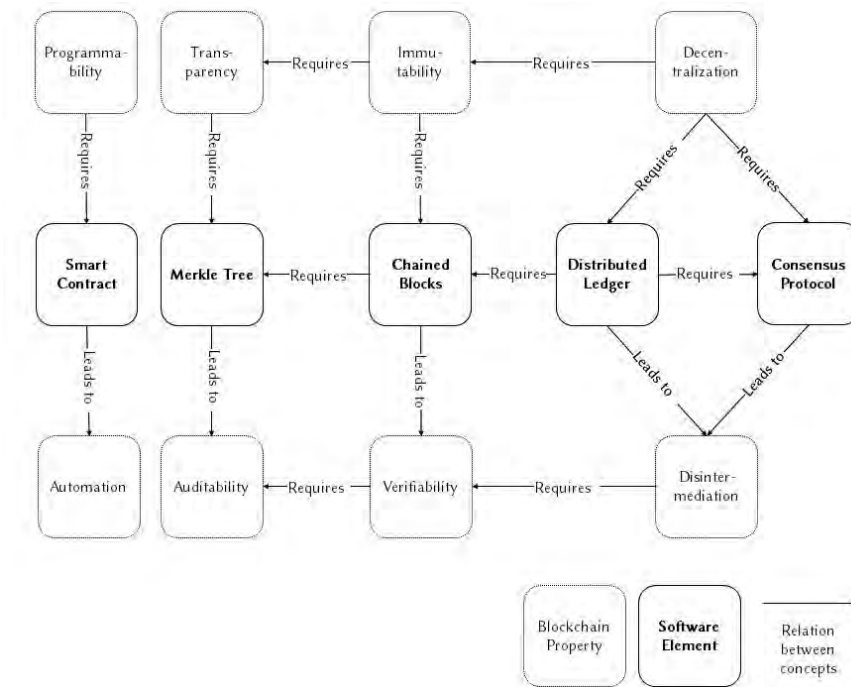
2.9.1 A Grounded-Theory of Blockchain Technology

The first step of in the data analysis procedure was to apply an open code each time the literature reflected a concept (see Sec. 2.3.3). A frequency analysis of the open codes (how often certain codes have been applied) unravels which concept are deemed important. Figure 2.8 depicts the top 10 most frequently used codes.

Blockchain challenges were mentioned most, followed by smart contracts, and consensus protocols. These results show that blockchain challenges are widely discussed in the papers under review and that the technology has not yet come to full fruition. The frequency analysis further showed that smart contracts are most frequently discussed, and can therefore be deemed a key concept for BCT future evolutions.

The third most often applied code is that of consensus protocols, and among them PoW. These findings resonate with the number of papers on the topic of PoW consensus protocols (see Sec. 2.3.5). Taken together, these findings show that literature is still predominantly focused on the PoW consensus protocol, whereas several other consensus protocols nowadays exist.

Surprisingly, the term permissioned blockchain is more often mentioned than permissionless blockchain which is not mentioned as one of the top ten concepts. However, these results can also be attributed to the perception that blockchains generally are public and



permissionless and as such that permissioned blockchains are an exemption to be specifically mentioned.

2.9.1.1 Grounding the Logical View of BCT

BCT encompasses several software elements that combined create the architectural properties of BCT. The results of the axial coding revealed what the intricate relation among software elements and properties.

The software elements of BCT work in concert to allow for secure transactions on a P2P network (see Sec. 2.5.1). We found that there is a strong dependency among these elements which has been depicted in Fig. 2.9 by the middle row of blocks: During a consensus protocol nodes verify transactions, which is not possible without the availability of a distributed ledger. In turn, the distributed ledger employs the concept of chained blocks that depends on a Merkle Tree to summarize the transactions. Because each software element is implemented to yield a particular property (see Sec. 2.7) these properties also are also connected. This relation is shown as the top row of blocks in Fig. 2.9.

Decentralization requires a consensus protocol and distributed ledger but in addition immutability of the transaction records. The immutability of transaction records is dependent on the degree of transparency of a ledger (see Sec. 2.7). However, transparency does not only guaranty the immutability of the distributed ledger, but also leads to auditability of the transaction records (bottom row Fig. 2.9). Verifiability of a blockchain requires auditability as without no proof can be provided that a transaction has not already been spend. An interesting observation is that programmability does not seem to have a direct relation to the other

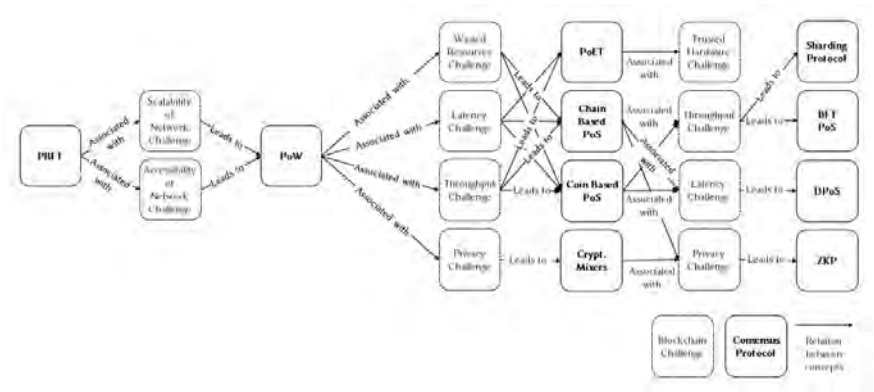


Figure 2.10: Consensus protocols related to challenges, in chronological order from left to right.

properties of blockchain. Every blockchain has a certain degree of programmability [274]. However, arbitrary programmability of transactions requires the concept of smart contracts which is optional for blockchain design [277]. The discussion of these results illustrates the complex relation among BCT software elements and properties. Considering these relations will be of importance to obtain the appropriate design when implementing BCT.

2.9.1.2 Grounding the Process View of BCT

Another relation that emerged from the axial coding process is the relation between BCT challenges and consensus protocols. In Fig. 2.10 this relation is shown simultaneously with the introduction of consensus protocols in chronological order.

Our results from the axial coding process show that each consensus protocol developed after the PoW protocol aims to tackle a particular BCT challenge. Furthermore, unsatisfied with prior protocols to address the challenges of BCT novel consensus protocols were introduced over time. For instance, the development of cryptographic mixers clearly aimed at providing more transaction privacy. To facilitate fully private transactions however, ZKP based protocols were introduced. While both approaches increase or facilitate fully private transactions these protocols were not designed to achieve faster throughput as compared to PoW.

On the other hand, PoS based protocols have aimed at improving throughput and latency performance as compared to PoW. Among the first of these attempts were the coin and-chain based PoS variants (see Sec. 2.5.3.4). As the performance aspects of these PoS variants were still considered unsatisfactory BFT based PoS, DPoS and sharding based protocols were introduced [158, 278]. Neither the PoS, DPoS or sharding consensus protocols address privacy of transactions.

Taken together the results show that there have been desperate attempts to address BCT challenges. A uniform approach that simultaneously addresses specific sets of problems is still lacking. This implication might have some important ramifications for future research efforts, for example, several research efforts by practitioners and scholars have led to consensus protocols that only pursue to improve one favorable property while a unified agenda to develop a consensus protocol that is designed with both privacy and performance aspects (throughput and latency) in mind is still lacking.

2.9.1.3 Grounding the Physical View of BCT

The grounded view describes the design patterns of a blockchain network. As discussed in Sec. 2.5.3 the design of the P2P network that supports the blockchain can differ. The frequency of the codes concerning BCT networks have been examined to establish a notion of their importance. What becomes clear from these statistics is that surprisingly permissioned blockchain networks are more mentioned than permissionless BCT networks. A closer examination of the papers in which the codes were used reveals that permissioned networks are predominantly employed to contrast the properties (negative and positive) of permissionless networks. Public networks are the third code that is most often applied, followed by private networks. These paradoxical networks are discussed in tandem to contrast their properties while Consortium networks are mentioned least. It must be noted however, that the terms consortium and private networks are often discussed in the same context.

The trends on BCT networks also sheds an important light on the developments in the field of BCT. These trends have been constructed by counting each time a paper was coded the network type in a given year. Thereafter the result has been divided by the total number of papers in the sample for a particular year. We deemed this last step appropriate to account for the fact that over time more work on BCT has been published. Analysis of the trends regarding BCT networks codes (see online materials in Appendix A.2 for more details) showed that in time more papers were mentioning private and consortium networks combined with permissioned models. When closer examining this trend we found that over time more papers were discussing private, consortium and permissioned networks. The literature under review [71, 88, 112, 211, 231] states that these network types were introduced as a response to the current challenges public oriented networks face (e.g. privacy and throughput). Therefore this trend can be explained by the fact that in time public blockchain challenges and limitations became more evident and thus alternatives were introduced.

2.9.1.4 Grounding the Use-Case View of BCT

The axial coding process of papers concerning BCT applications revealed that these applications have a distinct focus to which they have been categorized (see Fig. 2.5). A large strand of literature is focused on cryptocurrencies and improving the interoperability between chains using different coins, or fluctuations in the prices of cryptocurrency. The focus of these papers lies on *transactions* carried out using a blockchain or between chains and what influences these transactions. For these papers therefore the cryptocurrency itself becomes the *specific* focus of study as they are the embodiment of blockchain transactions. This contrast with other papers that see cryptocurrency as a means to an end and tend to have a more general focus on employing the technology as a whole. Several papers suggest the use of BCT to enhance the utility of IoT devices. Here, BCT *primary* serves to improve, amongst things, security and connections of these devices. In turn, IoT devices could potentially be used for a wide array of other applications. Discerning whether BCT is the application or rather a means to an end helps to categorize papers in the field of BCT.

Another observation resulting from the axial coding process is that the general application categories found in the literature sample utilize different properties of BCT. Record keeping applications mostly benefit from the decentralized nature of BCT. Whereas applications related to security and fraud detection seem to employ BCT for the sake of verifiability. For securities and insurance applications the auditability of the transactions is most important. Thus, besides categorizing applications based on their primary focus, one should also taken into account which of BCT's properties are predominantly utilized. This observation is important to take note of when developing BCT based applications as some properties can be

deemed more valuable for the design than others and can therefore traded-off against each other.

2.9.1.5 Grounding the Properties and Challenges View of BCT

BCT has been developed to attain certain properties yet also gave rise several challenges. We juxtaposed the results of a frequency analysis conducted using the codes applied for BCT properties and challenges to discuss their relation. The frequency of the codes related to BCT properties and challenges can be found in Appendix A.2.

Challenges related to privacy on a blockchain is mentioned most of all codes related to challenges. The immutability is the most frequent applied code related to BCT properties. Paradoxically, whereas privacy the most discussed challenge for BCT, the transparency of the ledger is mentioned as the second most important property of BCT. The decentralized nature of BCT is the single least applied code. Automatic execution is the least recurring code of related to a property of BCT. However, papers that predominantly discuss properties from the perspective of the Bitcoin do not mention smart contracts which is related to the concept of automatic execution.

A further analysis of the relation between the throughput and latency codes applied reveals that the coding of these two concepts coincides, resulting in an almost equal number of times these codes are applied. Governance of blockchain networks as a challenge is almost as frequently discussed as the technical issues of BCT (throughput and latency). The data storage and usability codes are less frequently used in comparison to the other codes concerning BCT challenges. Of these two codes usability is least often mentioned as a challenge. These results can be explained by the fact that most papers either focus on the applications for BCT or on architectural aspects without regarding the users perspective.

An analysis of the applied frequency of codes over time reveals the trends in BCT challenges. Again, the analysis has been conducted by counting each time a paper was coded a certain challenge in a given year. After obtaining the results these have been divided by the total number of papers for a particular year. The weighted number of codes per are challenge and year are depicted in Fig. 2.11 to provide a fine-grained perspective. The figure shows an overall decline between 2008-2019 in the mentioning of all challenges⁸. What is interesting however, is that when the codes are weighted (i.e. per year in relation to total number of papers) the results demonstrate that throughput and latency are the most important challenges to address. From publications in 2016 challenges related to BCT governance and usability first emerge. This shows that from 2016 onwards blockchain became mainstream adopted as the usability of BCT (for non-programmers) for the first time were discussed in [181, 261]. Moreover, in that same year governance emerged from the literature under review [78, 181, 209, 261] as a challenge. This literature discusses that novel updates to existing platforms were required due to several reasons yet that this turned out to be a challenge. In tandem, the governance issues from a legal perspective are discussed which means that BCT is was no longer regarded as a novelty but mainstream and a technology that needed to adhere to legal standards (see Sec. 2.8.5).

⁸For the years 2013 and 2019 only one paper has been included in the literature sample which imbalances the standardized weighted frequency of the codes for these years. Hence, in the scope of this analysis the results for 2013 and 2019 should be regarded as outliers.

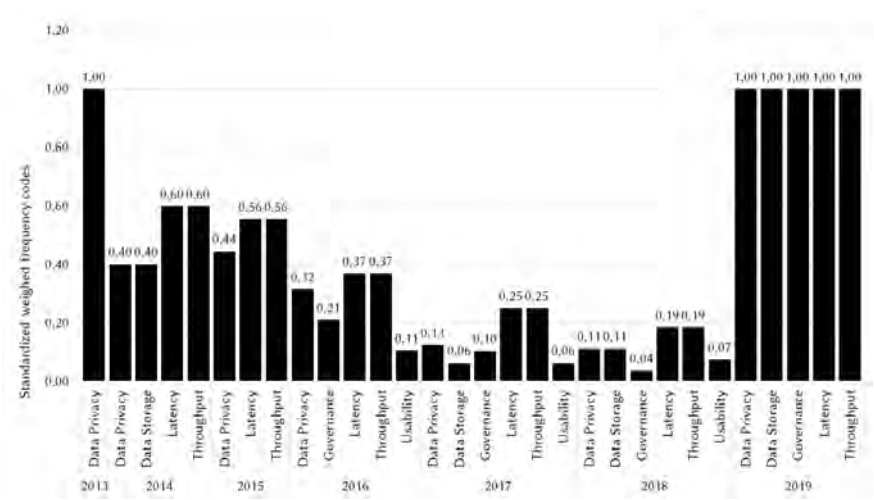


Figure 2.11: Trends in BCT challenges from 2008 to 2019.

2.9.2 Highlights and Observations

2.9.2.1 Chaining data using blocks

From the FCA conducted for this research we found that what sets blockchain apart from other forms of distributed ledgers is that transactions are stored in blocks. On the one hand, transaction data is stored in blocks to ensure the integrity of the distributed ledger. On the other hand, storing data in blocks also has its drawbacks.

Blocks have fixed data sizes and can only contain a limited number of transactions. Transaction that do not fit in the current block that is created have to wait to be processed and included in the next block. Given that both blocktime and blocksize for most blockchain networks are fixed these two parameters determine throughput capacity of a blockchain network [189]. The SegWit proposal (see Sec. 2.8) might partially solve this problem by expanding the size of blocks. However, to increase the throughput of public blockchain protocols further future expansions of the blocksize will be required. What the consequences are for the security of the network remains unknown however, because there has been no empirical investigation to test these configurations.

Nodes on a blockchain network need to keep a complete history of all transactions made on a blockchain network in order to validate them. Because the number of transactions that have been made on a blockchain is growing over time the data storage demands grow in parallel. Consequently, on the long-term it will be unsustainable for every node to keep the entire history of transactions [244]. Conversely, if only few nodes would be able to meet the store demands it would defeat the purpose of decentralization. A similar trend can be observed with regards to Bitcoin mining. Initially nodes mined blocks individually. However, due to increased mining requirements (CPU power) they eventually started collaborating in mining pools and mining became more centralized as a result. To address this problem some blockchains are utilizing the concept of checkpoints [158, 173]. Whether this solution aides in securing the network from attacks remains unknown. Another proposed solution is the use of lightweight clients. However, at least some nodes need to be full weight clients that keep track of all transactions thus, still have to burden themselves with storing large amounts of data.

Table 2.2: A complete reference over consensus protocols; instances are described along their characteristics, an implementation example and the source for argument of the claims.

| ID | Trust-level | Scalability (#Nodes) | Byzantine Fault-Tolerance | Throughput | Latency | Example | Source |
|-------------------|-------------|----------------------|----------------------------|------------|---------|--------------------------------------|----------------|
| PBFT | High | Weak | < 33.3% of faulty replicas | <2000 | <10s | Hyperledger Fabric v0.6 ⁹ | [44, 183, 285] |
| RAFT | High | Weak | <51% of faulty nodes | >10k | <10s | Corda ¹⁰ | [44, 183] |
| PoW | Low | Strong | <51% of computing power | <100 | >100s | Bitcoin ¹¹ | [183, 285] |
| PoET | Low | - | <51% of computing power | - | - | Hyperledger Sawtooth ¹² | [44] |
| PoS | Low | Strong | <33% of stake | <1000 | <100s | Tendermint ¹³ | [44, 183, 285] |
| DPoS | Medium | Strong | < 51% of validators | <1000 | <100s | Bitshares ¹⁴ | [183, 285] |
| ZKP ¹⁵ | Low | Strong | <51% of computing power | - | - | Zcash ¹⁶ | [72] |

2.9.2.2 Consensus in the wild

Besides the consensus protocols discussed in the results section (see Sec. 2.3.5) we reported more consensus protocols such as *Proof of Work or Knowledge* (PoWorKs) [21], *Proof of Vote* (PoV) [157], *Proof of Sequential Work* (PoSW) [60]. However, none of these protocols is supported by mature implementations, practical application, or empirical evidence of operational characteristics. For the sake of completeness, however, Tab. 2.2 offers an overview of all primary consensus protocols (without any derivation, e.g., PoSW with respect to PoW) and their claimed features in the respective literature. The table articulates every consensus protocol (ID in column 1) using (a) the trust-level required in the P2P network setting, (b) the scalability (i.e., whether the network can scale in terms of the number of nodes [124]), (c) the throughput, i.e., how many transactions per second the protocol can successfully process and (d) the latency, i.e., the time it takes to successfully confirm the transaction.

An interesting observation with respect to consensus protocols in Tab. 2.2 is that there seems to be a lack of systematic and empirical studies to test the claims around the proposed consensus protocols and their architectural properties. Of the papers reviewed, none based their statements about these properties and how they behave under different circumstance on the results of empirical testing. An exception is a study by Dihn et al. [72] that provides a framework to benchmark private blockchains. Public blockchains however, remain untested. Nevertheless some general observations can be made: With the introduction of PoW a shift is made from protocols that require high trust among nodes to a more trustless setting. This is not surprising given that one of the main aims of the original Bitcoin protocol was to make it expandable beyond a fixed number of authenticated nodes. Accordingly, an increase in byzantine fault tolerance of consensus protocols can also be observed. However, the results also demonstrate that an increase in the byzantine fault tolerance is coupled with a decrease in throughput and latency. For instance, the Tendermint PoS variant has a higher throughput and lower latency as the Bitcoins PoW protocol, yet it is also less byzantine fault tolerant (<33% to <51%).

For the PoET and ZKP proof based consensus protocols there is no data available with regard to their properties. Moreover, although it is stated in [53, 72] ZK-SNARK techniques incur large overheads in terms of storage space, there is little evidence to substantiate these claims.

The gradual shift from PoW towards PoS based consensus can have some important ramifications for the security of blockchains. With the exception of [158] little efforts have been made to investigate the unresolved security issues of PoS variants and how to resolve address them. One such problem is how to determine the deposit forgers are required to make to participate in the consensus protocol (see Sec. 2.5.3.4). If the gains of introducing invalid transactions outweigh the losses of a deposit the solution is will not be effective. Sharding of consensus seeking is a novel development mentioned in the papers under review. Yet we view that there are many unaddressed questions concerning blockchain sharding. When sharding a consensus protocol the nodes in the network will be distributed between several shards. How this distribution should take place remains unclear however. Provided that a secure manner has been found to distribute the nodes then it should be determined how many nodes must encompass a shard to make it secure as small shards (with few nodes) are easy to attack.

2.9.2.3 Blockchain Hybrids Emerging

Our literature suggests that more recently several interesting combinations of public blockchains with a permissioned model are under experimentation.

We observe from the coding process that the terms permissioned/private blockchain and permissionless/public blockchain are used interchangeably. Based on our extensive review, we propose that blockchain-oriented networks should be categorized based on both (1) network authentication coupled to permissions (permissioned/permissionless) and (2) accessibility of the network (public, consortium, private). Although the first blockchain network (Bitcoin) was public and permissionless, from the coding process it can be observed that in the papers under review permissioned and private/consortium blockchain networks are often mentioned (see online materials in Appendix A.2). One of the main reasons remarked in literature is that public blockchains have to utilize consensus protocols that can be regarded as slower than the ones that could be employed for private/consortium blockchains. Furthermore, having more control over the permissions each participant in the network has is another reason mentioned. We observe that private/consortium and permissioned networks are often regarded as a substitute to their public counterpart because of these challenges.

2.9.2.4 Mainstream Adoption of Blockchain Technology

An increase of academic works on the topic, and the overall increase of literature produced every year can be observed (see Sec. 2.3.5). These results show that BCT has been embraced by the mainstream. The usability of BCT for end-users and developers [176] could however, hamper the further adoption of BCT. Tools to develop blockchains/smart contracts seem to be absent. Especially for users that aim to employ smart contracts to define simple transaction logic a more simple and user friendly approach that does not involve programming would benefit the mainstream adoption of smart contracts and blockchain.

The results of this research show that governance of blockchain both in terms of the general ecosystem and at the platform level are a challenge currently. In many countries the use of BCT for applications or cryptocurrency has an opaque legal status. Making policy and regulation is difficult since a blockchain networks operate internationally and are not bound

to a single jurisdiction [185, 247]. Implementing updates or changes for a public blockchain network are a platform level governance challenge caused by decentralized decision making involving many participants. A solution to this problem might be the centralization of update permissions. However, that solution would be at odds with decentralized principles of blockchains and introduce several security hazards.

2.9.2.5 Safe Executability of Legally-binding Smart Contracts

Smart contracts are an important application for BCT as can be inferred from the number of times the code has been applied (see Sec. 2.9.1). All primary studies focus on discussing the security of Ethereum smart contracts. This trend is highlighted also in related work [5]. The most obvious implication of this shortcoming is that more rigorous, generalisable, and formalized approach to analyzing the safe and secure executability of smart contracts is required. A preliminary investigation of safe concurrent smart-contracts' executability Dickerson et al. [70] who describe an abstract solution to this problem without any rigorous evaluation. The trend towards sharding blockchain consensus can have implications for the execution of smart contracts what these exactly are remains unknown till date. Conversely, the word "contract" in the smart term definition contract would imply for its legally-bound enforceability under specific operational conditions [58].

2.9.2.6 Blockchain: Properties versus Challenges

BCT offers some interesting properties such as disintermediation, programmability, transparency, and verifiability that could potentially be beneficial for many applications (Sec. 2.7). However, the technology also has its drawbacks (Sec. 2.8). The work of Mougayar [185] provides a decision-making framework based on business parameters to aid in identifying the problems that BCT can streamline. Our work strives to provide a technical perspective to compound the business side explored by works such as Mougayar [185]. Gatteschi et al [88] suggest that when considering to adopt BCT one should ask: (1) Whether a shared database is required, (2) Multiple parties write the data, (3) Whether disintermediation is needed (4) and, whether it is required to see the linkage between transactions. In the similar vain Wüst and Gervais [272] argue that when deciding whether to adopt and design blockchain the contingencies depicted in Fig. 2.12 should be considered. They argue that if a trusted third party (TTP) always is available there is no need for a blockchain. Thereafter one has to make an analysis of the network participants to decide which type of blockchain network is most appropriate.

Besides the considerations presented by [88] and [272] from the results of this study we observe the following. First, a blockchain can disintermediate the transactions made on a P2P network between untrusted participants. However, as a result the throughput of most consensus protocols for blockchains is low and their latency high (see. Table 2.2). If these properties do not satisfy the requirements of the application these consensus protocols are not the most suitable solution. Visa for instance, handles 2000 transactions per second [193, 244] while most protocols currently cannot process transactions at that rate. Furthermore, when all network participants can trust each other a consensus protocol is not required. In such situations other solutions such as distributed databases [201] or P2P network [8] offer a less costly and faster way to exchange data or transactions. Second, not all blockchain platforms allow for the deployment of smart contracts which enable the programmability of transactions. Moreover, currently blockchain based smart contracts present some unresolved problems such as ensuring security [5] and legal enforceability [58]. In the past three decades several approaches have been suggested to develop contracts that are executed

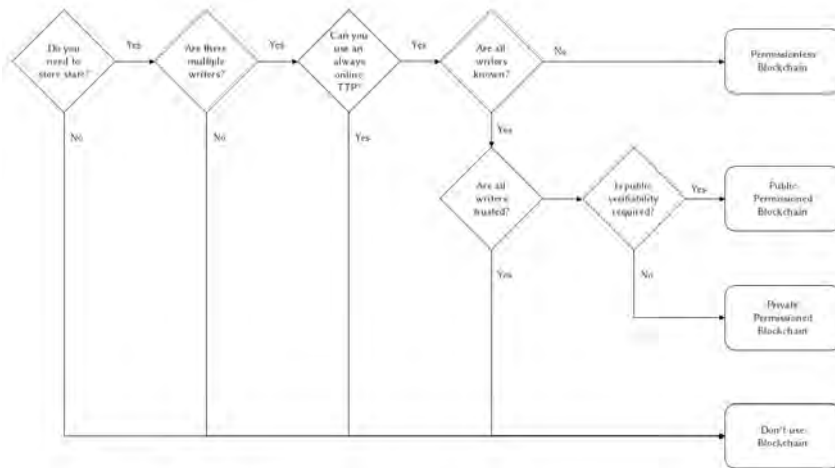


Figure 2.12: Decision-making model for blockchain networks; adapted from: [272].

electronically [138, 155] known as electronic contracts that currently have a higher maturity for these applications. Third, transparency and verifiability are a double-edged sword. In fact, these properties ensure the correctness of the distributed ledger, however transparency is not always desirable. More transaction privacy could be guaranteed by using ZKP's yet it is suggested that these incur additional overhead. An alternative is to perform the transactions on a blockchain network with a permissioned model. This could create censorship resistance which jeopardizes the security of the blockchain (see online materials accessible via a link in Appendix. B.1).

2.10 Research Gaps and Roadmap

The field of BCT is rapidly developing yet the results of this study point out four main research gaps that provide opportunities for future research. The research gaps and opportunities for future research have been identified using the results of the GT coding process (Sec. 2.9.1) combined with the descriptive (Sec. 2.3.5) and in-depth analysis of the papers under review (Sec. 2.9.2).

2.10.1 Consensus Protocols

Despite the fact that several publications discuss consensus protocols, few claims regarding the performance of these protocols are substantiated with evidence. One of the research gaps identified by this study is that evidence is lacking because the consensus protocols suitable for public blockchains remain untested. Similar to the approach presented in [72] for private blockchains, future research endeavors could *empirically test* the properties of consensus protocols for public blockchains. Addressing this research gap is especially important since the results of this study suggest that throughput and latency issues are still the major concern, and have a relative high urgency (see Sec. 2.9.1.5). Moreover, these empirical results aid practitioners making more informed architectural design choices. There is a trend towards developing consensus protocols based on *sharding* which could be further explored by future research as with the exception of [162] little work on this topic has been published.

In the same vein, a shift towards using PoS based consensus protocol for public blockchains can be observed (see Sec. 2.9.1.2). However, most published works on consensus protocols only specifically discuss PoW or provide a broad overview (see Sec. 2.3.5). Only three works [130, 158, 204] have been published on the topic of PoS, and only one of these [158] addresses the security of PoS protocols. More research on how PoS consensus protocols can effectively ensure the integrity of public blockchains is needed. Especially since there are still open issues to address such as *determining deposits* required for staking [158] during a PoS based protocol.

2.10.2 Data Storage and Privacy

Privacy has been shown to be a fundamental issue for BCT (see Sec. 2.9.1.5). As a remedy zk-SNARKs have been proposed (see Sec. 2.5.3.6 and Sec. 2.9.1.2). It has been claimed zk-SNARKs incur *large overheads* in terms of storage space. However, there is little evidence to substantiate this claim. Future studies could focus on investigating this claim and if proven to be true, investigate manners to decrease this overhead. Although zk-SNARKs facilitate private transactions in our sample no studies were found identified that investigate their effects on performance aspects (throughput and latency). Forthcoming research efforts could determine *the impact of zk-SNARKs on performance* which aids in further developing ZKP protocols in general. There is little work on how blockchains can become made *GDPR compliant* (see Sec. 2.8.4). However, we posit that like any other technology BCT has to operate within the current legal framework. The increasing amount of *data storage space* required by nodes is an unaddressed challenge. Some efforts have been made to address this problem such as the introduction of lightweight clients. The results of our study show that currently data storage is not perceived as a challenge with a high importance (see Sec. 2.9.1.5). Yet we posit that in time the problem might become more urgent when the throughput of consensus protocols will increase and in parallel, the amount of data required to be stored [244]. Thus, future research should investigate more efficient ways to store data, or determine if data can be omitted from the ledger.

2.10.3 Smart Contracts

From the results of this study *smart contracts* emerge as an important concept related to BCT (see Sec. 2.9.1). All of the works in our sample investigate smart contracts deployed on the Ethereum blockchain platform. Yet, more rigorous, *generalizable*, and *formalized approaches* to analyzing the safe and secure executability of smart contracts is lacking. Novel research could focus its efforts on ensuring the security and safety of smart contracts for platforms other than Ethereum. Literature related electronic contracts (e.g. [51, 140, 171]) could be a potential inspiration how to address the security issues of smart contracts. Furthermore, further work is required to establish the viability and implementation of *legally-enforceable* smart contracts. Given the trend towards sharding the consensus protocol research should be conducted how *cross-shard smart contract validation* can be performed.

2.10.4 Usability

This study has shown that multiple applications for BCT have been explored. However, the usability of BCT and smart contracts in particular remains limited since *user-friendly blockchain-oriented tools* are not widely available to non-programmers. The trends in challenges mentioned (see Sec. 2.9.1.5) suggest that over time this will become a more pervasive problem. Providing tools and approaches to ease the development of DApps or smart

contract could open up BCT to a broader audience. Therefore *user-friendly tools for smart contract* development would be helpful.

2.11 Limitations and Threats to Validity

Using the guidelines provided by Wohlin et al. [269] the limitations and threats to validity for this study were identified and are discussed in this section.

External Validity. First, developments in the field of BCT are introduced at a fast pace. Hence, some of these developments may exist but have not been published yet. Some papers on the topic are so novel that they have not been indexed yet and not included in the selected items. Another issue is that the terminology is still evolving and universal definitions for concepts such as BCT based smart contracts have not yet been formalized. This issue has been addressed by including search terms that are being used interchangeably in the search string (e.g. distributed ledger and blockchain technology) to ensure that all potential aliases of blockchain technology were covered. Items that were found using the search terms have been assessed thoroughly based on various dimensions of quality employing inter-rater reliability. Another threat to validity stems from including aliases of BCT is that findings of the study could also be related to distributed ledger technology. As a strategy to mitigate these threats we set out to define BCT based on a formal concept analysis using literature that specifically mentioned BCT in its introduction and background section.

Internal and Construct Validity. Second, to attain the results of the research questions a Glaserian-Straussian GT [93] Grounded-Theory coding approach has been used. Although an inter-rater measurement has been employed, the risk of observer bias is still present. Some additional codes were added to the list established during the pilot study. On these codes however, no inter-rater assessment has been performed.

Furthermore, the current body of knowledge on the topic of BCT to day remains limited. Many proposed consensus protocols in the field of BCT have not been rigorously empirically tested in terms of their properties. A prime example being the PoS consensus protocol for which a thorough assessment of the properties (e.g. throughput and latency) of its many variants is lacking in literature. In the same vein, with the notable exception of Li et al. [158], not much research has been carried out with regard to the security of the PoS variants.

In addition, in section 2.5.3 we have discussed several consensus protocols found in literature from a process view, indicating their process flow, idiosyncratic security issues and other properties. For the sake of space, only the consensus protocols that were recurring more than 3 times in at least 2 different papers across our primary studies were included in this study. However, as mentioned prior, this study has also identified other consensus protocols that were not discussed in full for space sake. Third, the findings of this research are partially based on grey literature sources. Inherent to grey literature is that the quality and accuracy of these sources can be disputable. In order to mitigate this threat we assessed each grey literature item obtained through our search strategy using multiple criteria based on the guidelines provided by Garousi et al. [86]. Furthermore, the assessment of the grey literature by the first and second author of this study has been subjected to an inter-rater reliability test (see Sec. 2.3.4).

2.12 Conclusions

This study was enacted to (1) analyze how blockchain technology can be defined, (2) provide a systematic overview of the state of the art concepts around that definition, (3) distill a grounded research roadmap around the topic. In section 2.4, using a Formal Concept Analysis (FCA) [175] approach, a systematic definition of BCT was distilled.

Beyond the operational definition above, using the well known 4+1 software architecture viewpoint framework [141], the architecture elements of BCT were fleshed out, specifically, our results recap: (1) the way a platform can be designed: (2) how transactions are processed and, (3) the architectural arrangements typically used for the P2P network underlying BCT. The third aim of this research was to flesh out what blockchain based applications have been discussed in the state of the art and how these can be categorized. The study reveals that there are three types of use cases for BCT: (1) cryptocurrencies, (2) smart contracts and (3) an array of more general applications which can be sub-categorized into five categories, namely, (a) security and fraud detection, (b) securities and insurance, (c) record-keeping, (d) Internet-of-Things (IoT) as well as (e) smart legal contracts. As a fourth objective of this study, we set out to determine the architecture properties of blockchain technology and their trade-offs. Data analysis reveals 8 coupled architectural characteristics — these properties are *trade-offs* exercised during blockchain architectural design. The fifth objective of this research was to identify the challenges for BCT. In the future, 6 main challenges for blockchain technology need to be addressed: (1) decreasing latency for the conformation of transactions; (2) increasing the throughput of transactions which is related to the design of the consensus protocol; (3) decreasing data storage requirement; (4) protecting the privacy of blockchain users; (5) data governance of blockchain networks; (6) the usability of the technology.

Finally, an analysis of the papers under review demonstrates that there are four research gaps that need to be addressed by future research concerning: (1) consensus protocols, (2) Data privacy and storage, (3) smart contracts, and (4) the usability of blockchain for end users. In our own research agenda, we plan to further analyse the results and data stemming from our study to further provide architectural and decision-making instruments for practitioners and academics alike. Furthermore, we plan to focus around the social and societal concerns around BCT, namely, its *privacy-by-design* [101] aspects as well as its end-user acceptance and maturity.

Chapter 3

Smart contracts

3.1 Introduction

Business processes that span organization boundaries pose a number of significant business and system level challenges [117, 203, 265]. Once the legal contracts between the trading partners are established, those should be monitored, enforced, and managed (including handling contract violations, termination, and update). As there exist often a lack of trust among the organizations, they face the delicate situation of trusting a specific partner for enforcing contractual obligations. To avoid relying on one trusted party, the business transactions need to be made transparent across partners. The transaction data including business interaction states, business objects and events should be shared among partners using a trustworthy, privacy-preserving, non-reputable medium.

Blockchain is an emerging digital technology that can execute and verify transactions between multiple parties without involving a trusted third party, and can record the transactions permanently [48, 72, 176, 188, 231, 262, 274]. It can provide a promising solution to address the contract enforcement and management challenges in untrusted business networks [176]. The two key enabling features of Blockchain are the distributed shared ledger, and computer programs (so-called smart contracts) that run on the ledger [117, 265]. The former can store immutable records of transactions in a peer-to-peer network of machines. Every allowed participant can access the data. The smart contracts are executed by all consensus nodes in the network. A participant can invoke the coded functions of a deployed contract by sending messages. The outcomes of the contract innovations are stored in the shared ledger, providing transparency to all relevant parties if conflicts arise. Note that while these computer programs are called "smart contracts", they are generally not very smart, and lacks the contracts to represent legal contracts [275].

Blockchain is gaining popularity in enterprises. According to Gartner "Hype Cycle for Emerging Technologies for 2018", blockchain will reach "Maturity" within 10 years. According to Gartner, by 2022, smart contracts will be adopted by more than 25% of global enterprises. There are already several blockchain providers such as Hyperledger, Ethereum, BigChainDB, and Kadena. Blockchain have promising use cases in areas such as trade finance, insurance, security industry, digital properties and rights management, organizational management, IoT, and energy [262].

The transparency and accountability enabled by the blockchain make it suitable to implement a (*logical*) trusted third party that can execute business transactions in untrusted business networks. The business transactions are much more complex operations than sending

This Chapter is based on a peer-reviewed publication in: Butijn, B. J., van den Heuvel, W. J., & Kumara, I. (2019). Smart Contract-Driven Business Transactions. In *Essentials of Blockchain Technology* (pp. 81-98). Chapman and Hall/CRC.

monetary resources from one party to the other [278, 285], and require enforcing the legal contractual obligations and rights of the involved parties [172].

In this chapter, we discuss the roles of the blockchain for enabling decentralized collaborative business processes across untrusted business partners. In particular, we present a meta-model that captures the key abstractions and constructs of smart contract driven business transactions. It maps the business-related aspects of business transactions to the system-related aspects that are necessary for executing transactions using the blockchain. We also describe the life cycle of smart contract driven business transactions, from the negotiation of legal contracts to the enactment of the transactions on a blockchain platform. To guide the implementation of the smart contract driven business processes, we also provide a reference architecture that supports our meta-model and life cycle model.

3.2 Motivating Example

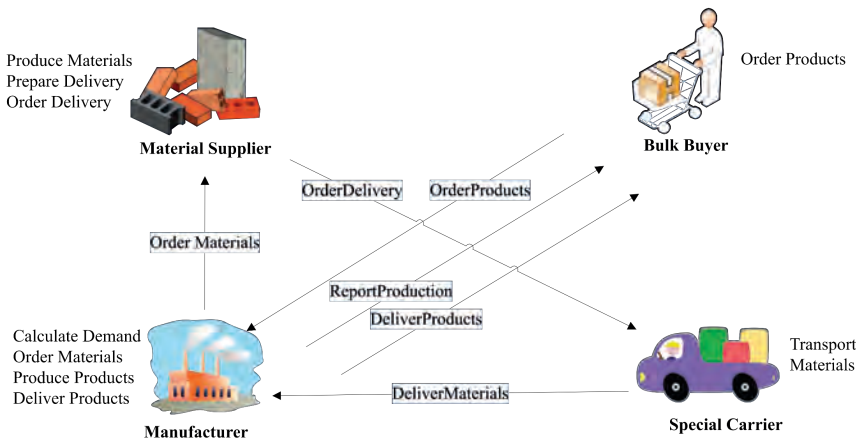


Figure 3.1: Supply chain example, Adopted from [265]

In this section we present a simple supply chain scenario (adopted from [265]) to motivate the smart contract driven the business transactions in the collaborative business processes.

As shown in Figure 3.1, the supply chain business network consists of four business entities: the Bulk Buyer, the Manufacturer, the Supplier, and the Special Carrier. Each entity can perform some business functions. For example, the Manufacturer can produce products, and calculate the demands for products, and the Supplier can produce raw materials. The supply chain business processes are often carried out as a set of multi-step business transactions. In a business transaction, an initiating partner requests the business functions from one or more partners, who perform the requested business functions, and may in turn request the business functions from some other partners. For example, the Bulk Buyer places a new order with the Manufacturer, which either accepts or rejects the order. If the order is accepted, the Manufacturer calculates the product demands, and orders the raw materials from the Supplier. Once the raw materials are ready, the Supplier asks the Special Carrier to transport them to the Manufacturer, which produces the products, and delivers the produced products to the Bulk Buyer.

The business transactions (and thus processes) are governed by the legal contracts between the business entities. Typically, the contracts express the obligations of the contract parties

to each other, and outlines services provided, business interactions allowed, performance, resources, conditions of providing services, and handling of contract violations. For example, the contract between the Bulk Buyer and the Manufacturer can state that the Manufacturer must deliver the requested products within 7 days after the order.

In business collaborations, as there often exist lack of trust between parties, the resolution of the conflicts between them can be challenging. Consider the case that the Buyer receives the products that do not match the requested product specifications, and thus refuse to accept the products. As the product are tailor-made for the Buyer, and cannot be sold to another buyer, the Manufacturer argues that the products are exactly matched with what were ordered. To resolve this conflict, the business interactions between parties must be made transparent while preserving privacy and security constraints, so that the malicious behaviors of parties can be readily spotted and legally penalized. Blockchain can help to build a trusted coordination environment for automating business collaborations between the untrusted business entities.

3.3 Lifecycle of Smart Contract Driven Business Transactions

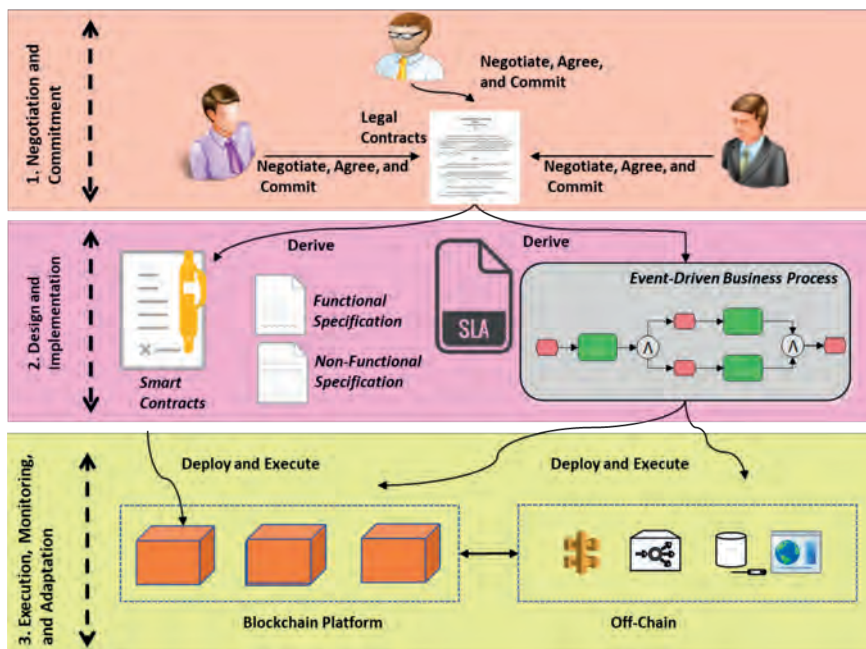


Figure 3.2: Life cycle of smart contract driven business transactions

Figure 3.2 depicts the life cycle of a business process running on top of the blockchain. It consists of three main phases: 1) negotiation and commitment, 2) design and implementation, and 3) deployment, execution, monitoring, and adaptation. The first phase considers the business related aspects, and the latter two phases consider the system related aspects.

In the negotiation and commitment phase, the participating business entities need to negotiate and agree upon the services provided and required, the terms and conditions of

service provisioning and requesting, and Quality of Service (QoS) requirements. Next, the legal contracts (documents) are created based on the mutual agreements, and the created contracts are signed by the relevant partners. The contracts generally express the rights and obligations of the contract parties to each other, and also outlines the guidelines for handling contract violations, and the conditions of commencement, continuation, and termination of the contract.

In the design and implementation phase, the required system-level artifacts are created and/or (partially) generated based on the legal contracts and the information about the technical capabilities (e.g., service interfaces, data format and message exchange standards used) of the participants. The key artifacts include the functional and not-functional requirements for the overall business process and the partners, Service Level Agreements (SLAs) between partners, executable smart contracts (in the contract programming language of the target blockchain platform), and business process models. The requirements and SLAs drive the design of the business process models. The business partners offer their business capabilities as business services, and the process models express the coordination of these services to realize the desired requirements while respecting the service level objectives of all the involved parties. These coordination logics in the process models can also be implemented with the smart contracts [265].

In the deployment, execution, monitoring, and adaptation phase, the implementation artifacts are deployed on on-chain (i.e., a blockchain network) and off-chain infrastructures. The relevant users should be able to enact a business process by sending the application-level messages to the system. The choreography model can be used to execute the process, where the deployed smart contracts act as the trusted coordinator. The enactment of the process instances are monitored to detect potential contract violations, and to trigger the relevant processes for handling each identified violation. The dynamics of the partner behaviors and the computing infrastructures, and the changing requirements and physical legal contracts require adapting the running business process, including deployed smart contracts.

3.4 A Reference Architecture for Smart Contract Driven Business Transactions

This section presents a reference architecture that provides a top-down layered approach to the development of the smart contract-driven business processes and transactions. It is inspired by the Service-Oriented Architecture (SOA) reference architecture [202]. As shown in Figure 3.3, the proposed reference architecture consists of six layers: physical business network (or business domain), business process, business transaction, business service, smart contract, and computing infrastructure.

Similar to the SOA reference architecture, the layer 1 presents the business domains in an enterprise such as production, finance, and human resources. Each domain consists of a set of current and future business processes that implement the requirements of the domain. Multiple business partners bound by legal contracts collaborate to realize these business processes, forming a business network. Layer 2, the business process layer, in our reference architecture is also similar to the business process layer in the SOA model. It captures the core business processes in a business domain, such as order management, production scheduling, shipping, and inventory management in the production domain.

Typically, the business processes need to integrate and coordinate business functions from multiple partners across the business network into multi-step business transactions. For

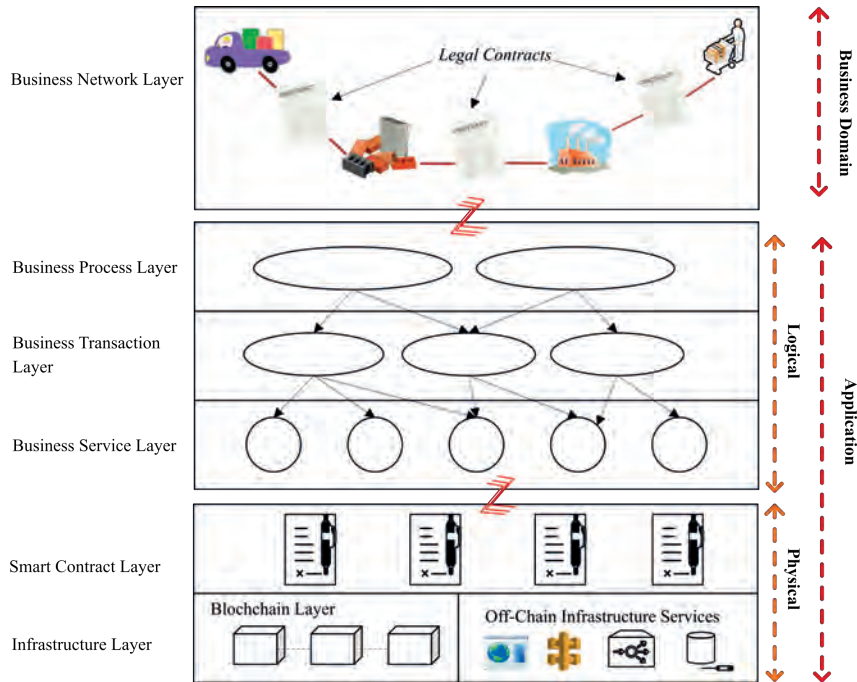


Figure 3.3: A reference architecture for smart contract driven business transactions

example, in the production business process, purchasing raw materials is a business transaction between the manufacturer, the supplier, and the special carrier. This transaction consists of steps such as place orders, produce materials, delivery materials, pay materials, and pay delivery. It may only be considered as successful once all raw materials are delivered to the manufacturer, and the relevant payments have been issued. Such business transactions can take days or even weeks. The legal contracts between the involved participants regulate these business transactions. Layer 3 in our reference architecture is to decompose a business process into a set of composable business transactions (reflecting the business semantics).

In service-oriented business environments, the participants expose their real world business capabilities as business (IT) services. Internally, a participant may realize its service as a complex business process that automates and coordinates its internal business tasks. For example, the manufacturer can provide a service that can be used by the bulk buyer to place orders, pay orders, cancel orders, and track delivery. Generally, the business services provide well-defined interfaces to hide their internal implementation details from other services (and thus other partners in the business network). A business transaction coordinates multiple business services into a logical atomic unit of work. The services interact with each other by requesting and providing service capabilities (business conversations). A transaction puts business constraints and invariants over these business conversations, for example, timing and ordering constraints on individual interactions. Layer 4 in our reference architecture includes the business services that can be coordinated to realize end-to-end business processes. Note that, in addition to the business services, utility (or commodity) services such as services implementing calculations and data processing algorithms are also necessary to implement business processes.

Business collaborations among untrusted parties require incorporating the transparency and accountability into the relevant business processes. The participants expect the contractual obligations and rights are duly enforced. They need to be able to share a trusted representation of the contract enforcement including the transactions and assets of each participant, and the decision making processes (e.g., handling of a contract breach or selection of a specific partner). With the blockchain technology, smart contracts can regulate the transactions using the business rules that the parties have agreed on. Ideally, the smart contracts should be derived from the relevant legal contracts. When the business interactions among participants are processed (as part of a business transaction), the smart contracts can apply the rules to check the compliance of the interactions, and produce the immutable records, reflecting the states of the individual interactions as well as the overall transactions and business processes. This shared trusted representation of the transaction data can give transparency to all relevant stakeholders if conflicts arise. Layer 5 in our reference architecture is to define the smart contracts for monitoring, regulating, and governing business transactions and processes.

Infrastructure services are necessary for automating and executing the smart contract driven business processes. We categorize them broadly into on-chain services and off-chain services. The former services are those provided by the blockchain platform to manage the life cycle of smart contracts, invoke the contract functions, store and access data, and so on. There are several blockchain platforms that support smart contracts, for example, Ethereum, Kadena, and Hyperledger.

The off-chain infrastructure services constitute the infrastructure services in the SOA reference architecture. Among them, technical services can provide the technical infrastructure enabling the development, delivery, maintenance, and provisioning of business services. They also offer capabilities to provide and maintain QoS such as security and performance. In this article, we assume that the choreography model is employed to coordinate the business services into business transactions and processes. Some of these coordination logics may also be realized by the smart contracts [55]. Infrastructure services also include monitoring and management services for monitoring the health and state of the business processes and resources, for detecting the potential contractual violations and trigger resolution policies, and so on. As the contract enforcement process should also be transparent to all parties, the smart contracts can also be used to realize it.

A smart contract can only access information that is stored on the ledger. However, some smart contracts operate in a wider ecosystem, and it might therefore be needed to acquire information about the outside world state and events. Moreover, the business services of the partners should be able to be triggered by the smart contracts, and vice versa [265]. This requires blockchain adapters/connectors that can receive messages or events from smart contracts and call external services, or receive service calls and send messages to smart contracts accordingly [265]. In the blockchain terminology, the trusted external data feeds are called *oracles*. Two current examples of oracles are Town Crier [284] and Oraclize [1].

3.5 Motivating Example with Smart Contracts

In this section, we present the realization of our motivating example using smart contracts. We first provide an overview of the case study design, and then describe some of the smart contracts used.

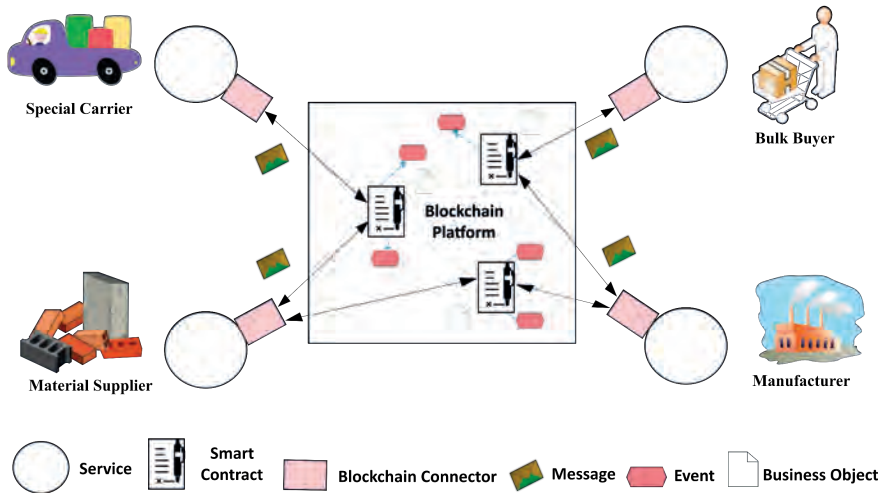


Figure 3.4: Motivating example with smart contracts

3.5.1 Case Study Design

Figure 3.4 shows the high-level architecture of the system. Each partner exposes its internal business processes as SOA services. For example, the service of the manufacturer offers the capabilities to ordering the products, canceling and revising an order, paying for the products, and tracking the statuses of an order. The implementations of these capabilities are internal to the manufacturer, and potentially use heterogeneous resources such as human workers, robots, software systems, and utility services. The blockchain-based smart contracts provide a trusted communication and coordination infrastructure for the business collaborations. The smart contracts between parties aim to implement the terms in the corresponding legal contracts. All the interactions (service requesting or transaction initiating interactions, and responding interactions) pass through the relevant smart contracts. A valid record of the each interaction is recorded in the blockchain as a blockchain transaction. The processing of an interaction by a contract rule may generate events and business objects. The services use a blockchain connector (e.g., the trigger component in [265] or a Web3.js based connector (<https://github.com/ethereum/web3.js/>)) to send interaction messages to the contracts, to read the business objects from the ledger, and to listen to the events generated. The detection of the event patterns can trigger the execution of the service operations (as in event-driven process chains).

Figure 3.5 illustrates the progress of a business transaction. The bulk buyer sends the product order request via a blockchain connector (client), which triggers the relevant coded function of the smart contract to process the interaction, and to record the state of the interaction (a transaction in Ethereum). The processing may generate events indicating the state of the interaction, e.g., *ProductOrderReq* event. The blockchain connector associated with the service of the manufacturer listens to this event type (or an event pattern), and execute the place order operation of the service. This service operation may trigger internal business processes for processing the order and deciding the next actions, for example, acceptance or rejection of the order, estimating the production demands, and ordering raw materials if necessary. Upon the completion of the place order operation, the manufacturer may interact with one or more partners, for example, notifying the order acceptance to the buyer, and placing a raw material order with the supplier. These interactions are also passed through

the relevant smart contracts (BB-MF and MF-SP). The smart contracts process and regulate interactions, and create a shared, trusted complete record of the transaction data, events, and assets, enabling the validation of the past transactions and the enforcement of the legal contracts.

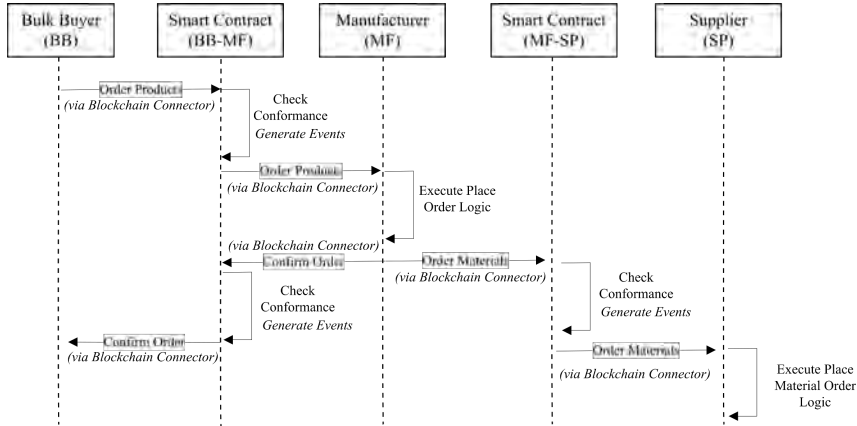


Figure 3.5: Sequence diagram illustrating part of a business transaction

3.5.2 Ethereum Solidity Smart Contracts

Let us consider some of the Ethereum solidity smart contracts used in the case study. All events, business objects, and the state of the contract function executions are recorded in the blockchain.

```

1  pragma solidity >=0.4.22<0.7.0;
2  contract BB_MF {
3    //addresses of the parties
4    address payable public mfAddr;
5    address public bbAddr;
6    //The Order business object
7    struct Order{
8      string goods;
9      uint quantity;
10     uint number;
11     bool init;
12   }
13
14   mapping(unit => Order)orders; //The mapping to store orders business
    objects
15   uint orderseq; //The sequence number of orders
16
17   //Event triggered for every new order from the bulk buyer
18   event OrderProductReqd(addresses buyer, string goods, uint quantity,
    uint orderno);
19   //Event triggered when the manufacturer sends the order acceptance
20   event OrderConfirmed(addresses manufacturer, uint orderno, uint
    delivery_date);
21   constructor(addresses_buyerAddr) public payable {
22     mfAddr = msg.sender;
23     bbAddr = _buyerAddr;
24   }
25   function sendOrder(string memory goods, uint quantity) payable public
    {
26     require(msg.sender == bbAddr); //Accepts orders just from buyer
27     orderseq++ //Increment the order sequence
28     //Store the Order Business Object
29     orders[orderseq] = Order(goods, quantity, orderseq, true);
  
```



```

30     //Emit the event
31     emit OrderProductReqd(msg.sender, goods, quantity, orderno)
32 }
33
34 function confirmOrder(uint orderno, uint delivery_date) payable public
35 {
36     require(orders[orderno].init); //Validate the order number
37     //Only manufacturer can confirm the order
38     require(mfAddr == msg.sender);
39     //Emit the event
40     emit OrderConfirmed(msg.sender, orderno, delivery_date);
41 }

```

Listing 3.1: Solidity smart contract between the manufacturer and a buyer (BB-MF)

Listing 3.1 shows a fragment of the contract BB-MF. It includes the addresses of the bulk buyer and the manufacturer. It also defines the business object *Order* and the events *OrderProductReq* and *OrderConformed*. The two functions *sendOrder* and *confirmOrder* are to intercept and validate the relevant service interactions. Listing 3.2 shows a fragment of the contract MF-SP. It binds the manufacturer and the raw materials supplier. The contract defines the business object *MaterialOrder*, the event *OrderMaterialReq*, and the function *placeMaterialOrder*. Each function generates the events and populates business objects as necessary.

```

1  pragma solidity >=0.4.22<0.70;
2  contract MF_SP{
3      //addresses of the parties
4      address payable public spAddr;
5      address public mfAddr;
6      //The material order business object
7      struct MaterialOrder{
8          string materials;
9          uint quantity;
10         uint orderNumber;
11         bool init;
12     }
13     mapping(uint => MaterialOrder)mOrders; //to store material orders
14     uint orderseq; //The sequence number of material orders
15     //Event triggered for every new order from the manufacturer
16     event OrderMaterialReqd(address manufacturer, string materials, uint
17         quantity, uint orderno);
18     constructor (address payable _supplierAddr) public payable {
19         mfAddr = msg.sender;
20         spAddr = _supplierAddr;
21     }
22     function placeMaterialOrder(string memory materials, uint quantity)
23         payable public {
24         require(msg.sender == mfAddr); //Only accepts order from manufacturer
25         orderseq++; //Increment the order sequence
26         //Store the Order Business Object
27         mOrders[orderseq] = MaterialOrder(materials, quantity, orderseq, true);
28         //Emit the event
29         emit OrderMaterialReqd(msg.sender, materials, quantity, orderno);
30     }
31 }

```

Listing 3.2: Solidity smart contract between the manufacturer and the supplier (MF-SP)

3.6 Discussion

In this chapter we discussed how blockchain enables decentralized collaborative business processes among untrusted business partners using a motivational example. The motivating

example delineates a standard process of a manufacturer handling an order. This is a rather common scenario in manufacturing.

Section 3.3 discusses the life cycle of a smart contract driven business transaction. This life-cycle on some phases coincides with that of a smart legal contract and also deviates on points. For instance, for smart legal contracts it is assumed that negotiation and commitment of parties is beyond their scope. The designing and implementation phase do roughly coincide with the akin life cycle phase of smart legal contracts. In the same vein, the executing, monitoring and adaptation phase are similar. However, smart legal contracts as we will demonstrate also have a dispute resolution and termination phase. These differences can be explained by the fact that smart contract used for business transactions only pertain business processes while smart legal contracts also need to cater for legal processes such as arbitration. A smart contract is a multi-purpose artefact that is deployed for a wide variety of use cases. Indeed, as we discuss in Section 3.2 smart contracts can be used for several business processes. This chapter only highlights one of these use cases.

An important consideration from Section 3.4 is that smart contracts are part of a larger architecture. This architecture enables trusted transactions between parties following a strict business logic that is represented in layers 2 to 4 with a particular notation. In case of a smart legal contract this logic is based on the clauses in a legal contract that needs to be represented in these layers. Since the logic stipulated in these layers also serves as the input for the design of the smart contracts, they link the legal and smart contract together. This also sets smart contracts used for generic business processes apart from smart legal contracts; When the smart contract is not used in a legal context, the business process, transaction, and service layers are designed based on any business process. Smart legal contracts on the other hand are based on the logic stipulated in a contract that is represented in these layers.

Another interesting observation to point out from Section 3.4 is that smart contracts form a self-contained system with the blockchain technology that it uses as the infrastructure underpinning their execution. However, it is not just the blockchain infrastructure that is connected to the smart contract. Some off-chain services such as Oracles might provide data to the smart contract as well. Effectively this makes the blockchain system less self-contained, but it also introduces several problems. Furthermore, the stipulation of the logic that a smart contract follows is predominantly specified at the business layer. In other words, the requirements for the development of the smart contract have to be obtained from the domain.

Ultimately smart contracts are code and using the case study in Section 3.2 we demonstrate how a process would be translated into a smart contract. This case study demonstrates that at least for simple business transactions, smart contracts can facilitate them. One of the limitations that can be noticed from this case study however, is that all actions need to be formulated as functions. Moreover, the majority of actions within the process will be performed by human agents and not by the contract itself. This highlights an important limitation of smart contracts.

3.7 Conclusion

The current chapter expounds on the concept of smart contracts using a case study. An important conclusion of the chapter is that smart contract are strongly tied to the infrastructure they are deployed on. Further, using smart contracts for business transactions will require another approach as compared to smart contract that are used for legal contracts. The life-cycle between smart contracts designed for business transactions and that for legal

contracts differs which has some important ramifications for their design. One of the advantages of smart contracts is their potential to enforce predefined logic. Blockchain platforms provide different languages to stipulate the logic of a smart contract. However, the manner to denote the logic within a domain is not specified. It is from that respective domain that the requirements for the development of the smart contract need to be obtained.

Chapter 4

Research Methodology

4.1 Introduction

In this chapter the choices made for the research paradigm and research design are discussed. Every research encompasses some assumptions stemming from the paradigm that the researcher has adopted, and follows a reasoning that has led to the selection of a specific research method. Researchers should make these assumptions explicit and argue the suitability of the method in light of the of the paradigm [198].

This chapter firstly presents and discusses the main research paradigm adopted for this research: Design Science Research (DSR). DSR originates from the engineering discipline and focuses on science of the artificial [232]. The DSR approach will reoccur throughout the dissertation as it is an artefact development centered paradigm that is well-aligned with the purposes of this study. After discussing design science research itself, it will be argued why design science has been adopted as the primary paradigm for this research.

While we argue that DSR provides a suitable lens to study the development of novel artefacts, the paradigm does not prescribe a method to build an artefact and evaluate the outcomes of the design process. We take on the call from Von Alan et al. [260] that existing knowledge should not be ignored. Seeking to leverage the current knowledge base on software development, this research sets out to design an artefact (method) that caters for the coding of smart legal contracts based on the requirements stated in Section 1.5.2 using the MDA philosophy.

Adopting the MDA philosophy to software development has several advantages such as increased productivity, portability of code, interoperability between platforms, maintenance of the code, and the models themselves foster communication between stakeholders [37, 118, 133]. However, to attain these advantages a specific method must be followed. This chapter will further discuss what using the MDA approach for the software development entails. Models used for MDA need to be written in a language that caters this purpose, the chapter will discuss how these languages are used. Further, a more detailed explanation is provided how mapping enable the transformation process.

The advantages of using the MDA philosophy connote directly to some of the requirements we have defined for the design of a method to support the creation of smart legal contracts. Employing MDA requires the creation of several models that have shaped the methods employed to conduct this research. These methods and the design for this research will be made explicit and discussed.

In this chapter we will proceed in the following manner: In the next section (4.2) we will discuss DSR paradigm and it's suitability for this research. To provide an understanding of MDA, we provide a further explanation of the foundations underpinning the philosophy. First we

will discuss how the three viewpoints of MDA are linked to different modeling levels in 4.3.2. Next, thereafter in 4.3.1 we will discuss the languages available to denote the models. In subsection 4.3.3 we discuss how models are transformed into code using mappings. Some important ramifications of adopting the MDA approach to software development are discussed thereafter in Section 4.3.4. The rationale of the research design will be discussed in Section 4.4. Thereafter, in Section 4.5 we will discuss the benefits against considerations of MDA as the philosophy underpinning the design of the artefact. The chapter concludes in Section 4.6.

4.2 Research Paradigm

Most information systems studies are characterized as behavioral science or design science. The behavioral science paradigm seeks to develop and verify theory. By developing novel and innovative artefacts the design science paradigm aims to enhance both human and organizational capabilities [260]. Design Science Research, pioneered by Simon [232], supports a logical research model that advocates the design of innovative artefacts that solve real-world problems. Within DSR artefacts can include but are not limited to models, methods, constructs, instantiations and design theories, and social innovations. Employing DSR to develop an artefact like a method designed for this research would therefore be suitable. Such an artefact is perceived to contain knowledge ranging from the design logic, methods of construction, and tools to information related to the environment in which the artefact will operate.

Within design science research a strong emphasis is placed on the relevance that the artefact has to the application domain. The emphasis on the aspect of design science to cater for, or enhance the efficacy of artefacts in context of real-world problems sparked a debate on what the nature of design science is. In their seminal paper Von Alan et al. [260] address this issue and argue that design science is most suitable for wicked problems. Wicked problems have the characteristic that there are unstable requirements for their solution, flexibility in the design process is required, complex interactions among sub-components exist, and developing an adequate solution relies on creativity [41, 218]. Given the rapid pace of legal and technical developments in the field of BCT the requirements for the artefact might be change. Flexibility is consequently required to face the changing demands from the environment. Furthermore, there exists a complex interaction between sub-components that stem from two different disciplines and that are reciprocally affected by their respective impediments. Taken together, these reasons constitute to another argument to choose an DSR approach over others.

Hevner et al. further suggest that design science should address wicked problems through the development and rigorous evaluation of an artifact in its context. According to the DSR paradigm an artefact is constructed based on requirements stemming from the environment and current knowledge base. Thereafter the artefact is evaluated using these requirements and measuring performance measures. In the same vein, Peffers et al. [206] argue that an artefact developed using a design science must be: (1) developed to address a problem, (2) relevant to an unsolved business problem, (3) rigorously evaluated on quality and efficiency. However, design science as a paradigm provides little guidance on how to concretely conduct the research process as it is not a methodology [23]. To address this need Peffers et al. [206] proposed the Design Science Research Model (DSRM). The DSRM encompasses six consecutive steps that are depicted in Fig. 4.1.

The first step is the identification of the problem and motivation to solve it. A second step is to define the objectives the artefact tries to solve, these must be in line with the problems

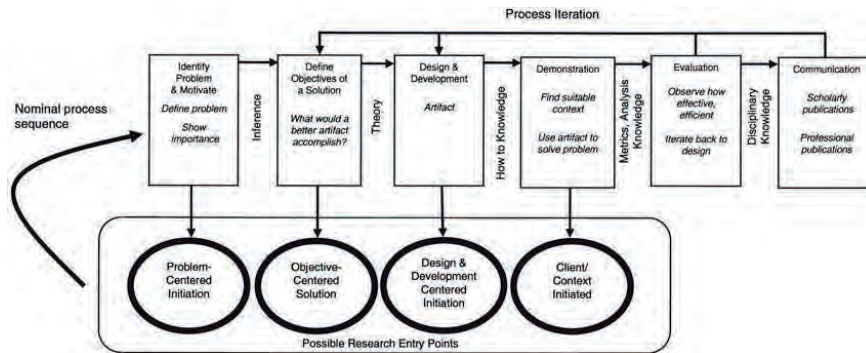


Figure 4.1: Design Science Research Model (DSRM)

identified in the first step. Based upon these objectives and the current knowledge base the artefact is designed and developed thereafter. A fourth activity is the demonstration of the artefact. This can be done in many manners, yet showing instances of the artefact is most common. Thereafter the design of the artefact can be evaluated against the objectives formulated in activity 2. The design and outcomes of the artefact can thereafter be communicated. Researchers employing the DSRM are not expected to go through all six steps sequentially because it is possible to initiate research at almost every step and move towards the last step. There are four potential starting points when employing the DSRM: (1) problem-centered initiation, (2) objective-centered initiation, (3) design & development centered initiation, and (4) client/contract initiated. For this research we initiate the design cycle from a problem-centered perspective by first fleshing out the problems and clarifying their definition.

Wieringa [266] proposed a design cycle with the aim of further specifying the steps of the DSRM and framework of Hevner. The design cycles proposed by Wierenga aids in bridging the gap between the DSRM and the framework proposed by Hevner by including specific knowledge questions on the artefacts' design and evaluation. Wierenga's cycle encompasses four steps problem investigation, artefact design, artefact validation, and when the artefact satisfies the requirements, implementation. Note that when the artefacts do not satisfy the requirements, the cycle will be re-iterated until satisfactory. As can be inferred from the work by Wierenga and Peffers et al. from the perspective of DSR the design process of an artefact is not linear but rather iterative.

The DSR paradigm is a lens often employed in the field of software engineering to develop software artifacts [266]. As mentioned, DSR does not prescribe a specific method or approach to develop software. Software development is a process that costs valuable resources and time [77]. To ease the process of software development practitioners and scholars alike have suggested various approaches to ease the development of software. One of these approaches proposed by the Object Management Group is that of MDA [27] that will now be discussed.

4.3 Model Driven Architecture

The cornerstone of the MDA philosophy is the usage of models to abstract the viewpoints of a system required for development with corresponding models. Another central aspect of MDA is the concept of model transformation, in which one model is converted into another

model of the same system. The transformations between the models allow the developer to develop a system from an abstract viewpoint and materialize these viewpoints into code [228].

4.3.1 Modeling Levels

Software systems are a set of elements grouped together to attain a particular goal. A description and specification of a system, and its environment can be viewed from various viewpoints. Viewpoints are a technique for abstraction that use a select set of architectural concepts and structural rules to address specific concerns about that system [37]. The MDA approach specifies three viewpoints on a system: a computation independent, platform independent, and platform specific viewpoint [27]. These viewpoints are hierarchically organized from more to less abstract respectively. Each viewpoint requires at least one model that specifies the concepts required to stipulate the viewpoint. Models are a reduced representation of a software system that portray the properties of interest and their environment from a particular viewpoint, that are denoted graphical or textual [37]. The models required for software development following the MDA philosophy include, a Computation Independent Model (CIM), Platform Independent Model (PIM) and a Platform Specific Model (PSM) for each platform [27]. If needed, each viewpoint can be further specified using additional meta-models. All of the models have a specific purpose within the MDA philosophy that is portrayed in Figure 4.2.

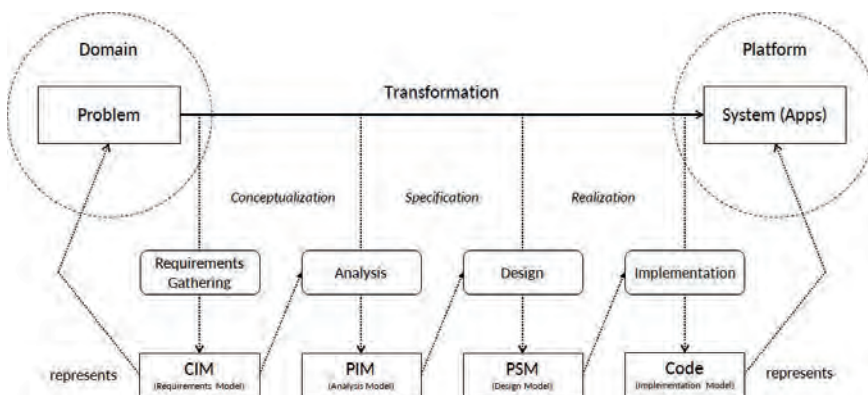


Figure 4.2: Use of different models within MDA. Inspired by [233]

A computation independent model focuses on the domain of the system, and stipulates concepts and requirements for the system. Concrete details on the structure and processes within the system are not specified from this viewpoint. This is not required as the main purpose of a CIM is to encompass any relevant knowledge or to gather requirements from the domain. Domain ontologies and business models are often employed to define a CIM with a vocabulary that is familiar to domain stakeholders [87].

The aim of the platform independent model is to view the operations of a system in a platform agnostic manner. Platform agnostic means that the viewpoint only pertains a specification of the system that does not differ between platforms. The PIM is constructed in a fashion where commonalities and differences between platforms are laid bare.

Platform specific models incorporate the platform independent viewpoint, and further specifies deployment of a system for a specific platform. A PIM needs to adhere to any architectural standards and constraints that the platform for which it is created specifies. The model

provides a set of technical concepts that represent the distinct parts that the platform encompasses, and the services that are provided by the platform. In conjunction with the PIM it is a vital part of MDA as it allows for the translation between a platform agnostic context to a platform specific specification [37].

The ultimate goal of stipulating these models is to enable the generation of code. Albeit that a PSM enables the specification of the requirements of the system for a platform, it does not generate the code itself. Cartridges or code generators "write" the code based on the specification of the PSM.

4.3.2 Modeling Languages

The MDA philosophy revolves around a meta-modeling architecture that encompasses four layers and additional standards that are all supported by a formal language [133]. In Figure 4.3 the hierarchy of these layers is depicted. Elements within a lower layer are instances of an element within a higher top layer. Effectively, higher level layers models can be used to specify languages at a lower level.

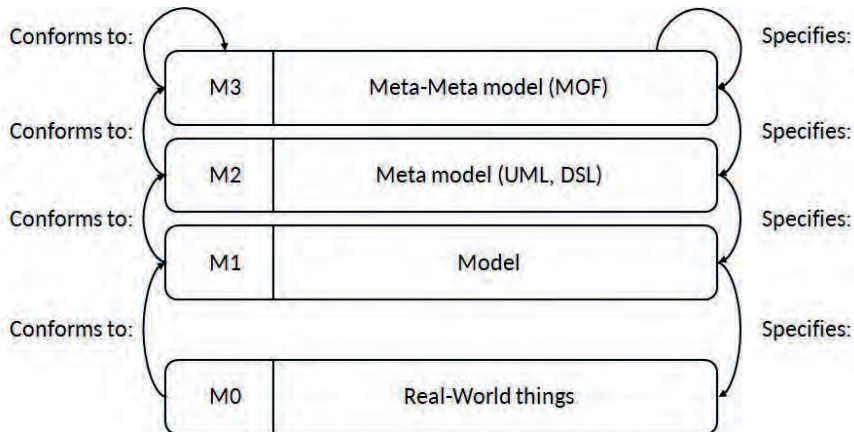


Figure 4.3: Hierarchy of metamodels and languages. Based on [144].

The M3 level of the meta-meta-model is the highest level within the architecture. MDA advocates the Meta Object Facility as the language for this layer. Meta Object Facility (MOF) is an abstract language that is self-defined to support the specification, construction and management of technology agnostic meta models [37]. It provides a foundation for the definition of other meta modeling languages such as the Universal Modeling Language (UML) [220] or itself. At the M2 meta model level the elements allowed within a model at a M1 level are defined. The UML is a common language to define these models that encompass classes, attributes and associations amongst them. In other words, metamodels are models of languages employed to define other (modeling) languages. Modeling languages are defined by a metamodel and specify all possible model variants that are conform to that meta model. Elements included in models at the M1 level are instances of elements of the M2 level. Models on this level include abstract elements such as classes (e.g., Person) and their attributes (e.g., Name). The M0 level pertains the reality where there are concrete instances of models at the M1 level such as objects in a programming language [15].

The models at the different levels must be written in a well-defined language. However, CIMs can be modelled in any language. Given that the aim of a CIM is to gather domain related

requirements a Domain-Specific Language (DSL) is often employed [87]. A domain-specific language is a language that is developed for a particular domain, context, or organization to simplify the task of domain practitioners that are required to describe things within that domain [255]. Examples of a domain-specific languages include HTML for the internet, SQL for database queries. Some domain-specific languages have as a goal to model this knowledge, in which case they are also called domain-specific modeling language. A prominent example of a domain-specific modeling language is the Business Process Model Notation (BPMN) that is used to model processes within and between organizations [195]. Domain ontologies are often specified in the Web Ontology Language [239]. However, the Web Ontology Language is designed for knowledge representation and is not suitable to build systems, while UML does support the construction of software systems [227]. Moreover, UML is machine readable, and if used in conjunction with the object constraint language allows for the specification of first order logic. Therefore, UML is sometimes used to define an ontology [87].

Contrary to a CIM, a PIM and PSM are modelled using a formal modeling language, for which the UML is the standard proposed by the OMG [197] and the de facto standard in practice [118]. UML is a general-purpose modeling language for object oriented-oriented modeling that can be applied across multiple sectors and is not restricted to one domain. When using UML several aspects of a system are portrayed with activity, use case, sequence and class diagrams. Class diagrams aim to depict the classes and their attributes of things that are instantiated as objects. Although UML is a versatile language, it can only partially express the information needed to define a modeling language. In fact, most general-purpose modeling languages only cater for the definition of simple modeling constraints of novel languages. Oftentimes these are basic cardinality constraints which restrain the number of associations between elements within a model [37].

To support a more elaborate specification of constraints the Object Constraint Language (OCL) was introduced [264]. The OCL is a formal general-purpose language that is standardized for MDA by the OMG. OCL complements models with additional textual rules to which these models must adhere. It is typed meaning that every expression in OCL has a type and complies to the rules and processes of that type. The syntax to define a constraint in the OCL is always written as follows:

```
context <classifier>
    inv <constraint name>:<Boolean OCL expression>
```

The context defines the class to which the constraint applies, the optional constraint name serves to identify the constraint, and the Boolean expression that must hold true to satisfy the constraint. When the OCL is stipulated it does not create any side-effects such as modifying the state of the system, it can only constraint said state. Related to this property, expression about these constraints are declarative in nature. The definition of OCL does not include any specific implementation, and requires a tool to be used. Hereunder we provide a brief example of a constraint denoted in OCL.

```
context Meeting
    inv EndBeforeStart:self.end > self.start
```

In the example the class meeting has the attributes (indicated by the keyword `self`) `end` and `start`. Logically, it does not make sense that date and time the meeting ends is before the start date and time of a meeting. The constraint above guards against such errors by stipulating that the value of the `self.end` attribute must be larger than the `self.start` attribute. Boolean expressions in OCL do not only have to contain operational operators, the OCL provides a wide array of keywords such as `implies` or `if` statements that enables the definition of fine-grained specifications of a constraint.

4.3.3 Mappings and Transformations

A central aspect of MDA is the process of model transformation, during which a source model is converted into a target model of the same system [228]. Transformations therefore in essence effectuate the translation between a source language and a target language. The transformation process between the models allows developers to develop a system from an abstract viewpoint, and materialize these viewpoints into code. Model transformations in MDA focus solely on conversion from a PIM to a PSM and vice versa. The literature on MDA remains silent on transformations from a CIM to a PIM, and PSM to code [235]. Figure 4.4 depicts the translation process and how it is related to the MDA architecture.

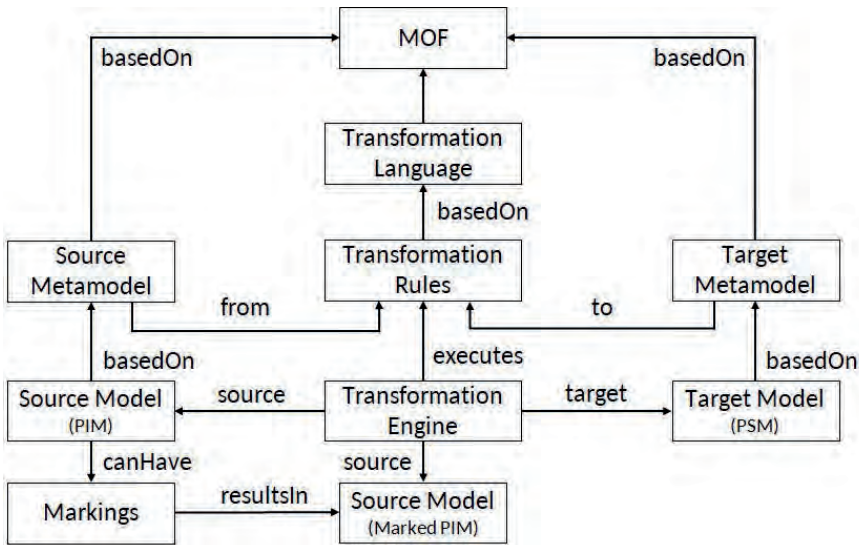


Figure 4.4: Model transformations in MDA. Based on: [104]

A transformation between a PIM and PSM is enabled by employing a transformation engine that executes the transformation following mappings. Mappings¹ are a specification of the transformation rules, constraints and other information required to transform a PIM into a PSM. The OMG provides a standardized model transformation language that fits within the Meta Object Facility environment called the Query View Transformation language. Another commonly used transformation language is Atlas Transformation Language (ATL) [120]. Each mapping specifies the correspondence between one or multiple elements in a source model to the target model. Three categories of mappings can be discerned: (1) one-to-one mappings, where an element in a source model corresponds to another element in the target model, (2) a mapping of the one-to-many kind specifies the correspondence between one element in the source model to several elements in the target model. (3) Many-to-one mappings denote how multiple elements in the target model are translated to one element in the source model [37, 160]. To exemplify how mappings are specified in the ATL please consider listing 4.1 that denotes a simple one-to-one mapping.

```

1 module Person2Contact :
2   create OUT : MMb from IN : MMA ;
3
4   rule Boss2Contact {

```

¹Mappings are sometimes referred to as transformation rules, and unfortunately the terms are often used interchangeably creating unnecessary confusion [104].

```
5
6  from
7      p: MMA!Person(
8          p.function = 'Boss'
9      )
10 to
11     c: MMb!Contact(
12         name <- p.first_name + p.last_name
13     )
14 }
```

Listing 4.1: Example of a mapping in the ATL

The example in listing 4.1 specifies how an object of the `Person` class is transformed into an instance of the `Contact` class. To enable this transformation a specific module, in this case `Person2Contact`, is used that inspects `MMA` (Meta Model a) and transforms it to elements in `MMb` (Meta Model b). The keyword "rule" indicates the start of the transformation rule, followed by the name of the rule (`Boss2Contact`). For the transformation a parameter (in the example a `p`) is used to capture the object. In this example only objects of the `Person` class are transformed into a contact when they have the function 'Boss'. The "to" keyword specifies the targeted element, in this case `Contact` that has a property called `name`. This property is created by combining the `first_name` and `last_name` properties of the `Person` object as indicated with the arrow.

In some instances, it is not possible to stipulate mappings that fully automatically and directly transform elements in the source model to the target model without additional knowledge. To remedy this problem, in practice two types of mappings are used [144]:

1. Model type mapping that defines the mapping rules at a model element level. Mapping rules of this type define how elements in a PIM are directly translated into elements in the PSM.
2. A model instance mapping stipulates how elements within a model are specifically transformed using user provided marks. The markings indicate the user choice for the translation of certain elements.

Marks are applied by a user to elements within a PIM to indicate in which manner the element in question is transformed. Elements in the PIM may be marked multiple times by disparate mappings, and as a result are transformed according to each of these respective mappings. The marked or unmarked version of the PIM is thereafter transformed to a PSM and a record of the transformation is generated [37]. A record of transformation includes a map from each element of the PIM to the corresponding elements of the PSM, and shows which parts of the mapping were used for each part of the transformation. The main purpose of a transformation record is to provide *traceability* between the inputs and outputs of the transformation process, and *transparency* on the transformation process itself. The models themselves are captured using an intermediate representation such as XML or JSON [228]. Ultimately the models need to be transformed into code, these types of conversions are referred to as model to text (read code) (M2T) transformations. Generating code from models is achieved by a code generator or cartridge, however templates can be employed to specify how the code should be rendered. Templates define a specific type of transformation that utilizes parameterized models, or patterns of elements within a model to complement or substitute model type mappings and model instance mappings.

4.3.4 MDA: Benefits and Considerations

Models themselves do not make for transformations and require an architecture such as the one discussed in subsection 4.3.2 for this purpose. Once the models and the system to support the transformation have been built practitioners can enjoy the increased productivity and ease of code creation that the MDA approach enables. While acknowledging the benefits of MDA some actually important considerations of the MDA philosophy have to be taken into account. Although MDA offers many benefits for software development, in parallel there are some important considerations that are associated with these benefits.

Communication – The use of models for MDA aids in bridging a communication gap between requirements analysis and implementation. As discussed in subsection 4.3.1 models are meant to represent a certain reality and can be used by domain experts to communicate concepts and relations relevant to their respective domain. Models have shown to be a viable solution to provide a lingua franca between stakeholders from business and IT [37]. This translates to bridging the gap between the needs within business and IT realization. However, a model is a simplified representation of a certain reality. Thus, models might not be able to fully capture the complexity and particularities of a domain or reality [144]. On the flip side, this is perhaps actually one of the strengths of models. It is because models present a simplified view on a reality that allows for the creation of workable and real solutions [37].

The language used to describe a model strongly determines what can be modeled and how [144]. Some researchers have criticized UML for being too simplistic and lacking expressiveness to really capture real-world complexity. In contrast, others [103, 235] observe that UML has become a complex language that requires an experienced modeler to create sound and workable models. Not all domain experts possess this expertise besides that of their own field. Notwithstanding this criticism, the use of the UML to draw models has several advantages and is the most commonly used and understood language among programmers [118].

Portability – The portability of the models and code is another benefit that MDA provides. Because a PIM is a platform agnostic and abstract representation of a system, the model and specifications stipulated therein can be reused for several platforms. The platform agnostic approach makes defining shared requirements for each individual platform redundant on an abstract level [118, 133]. Only at the PSM level more specification is needed for individual platforms. Hailpern and Tarr [103] argue that while abstracting a viewpoint into model might reduce complexity of a domain, in concert it might only move complexity to another level because several artifacts are now needed to create the instance of that domain. They argue that therefore using models to create software does not really reduce complexity, but rather moves it.

Productivity – MDA promotes productivity by advocating the use of standardized models. Once these models have been created the models and coherent code can be reused. Therefore, writing custom code for each application becomes redundant. More important, the models themselves can be reused for identical or similar applications often with little adjustments lowering the cost of software development [118, 133]. However, initially the upfront investment costs to achieve the desired gain in productivity might be high. Researchers [103] have argued that initially more expertise is required to make the models as compared to traditional approaches of software development. The kind of expertise needed might also be different, and thus increase the number of experts involved. Besides this upfront investment in MDA each viewpoint needs its own specification language and a system to execute this language creating redundancy in the process. Subsection 4.3.1 and Subsection 4.3.2 substantiate this point: for the creation of an architecture that supports MDA a plethora of languages are required and several tools.

Interoperability - The CIM and PIM models ensure that the relations between PSM's for different software platforms are linked by bridges so that the concepts of a PSM specified for one platform can be used for the PSM of another platform [133]. Achieving a model transformation from and between models might be rather difficult. When MDA was initially introduced this problem was further magnified by the fact that few suitable tools were available to support the MDA philosophy [43, 235]. Although this problem has been remedied by the introduction of tools like Eclipse not all problems are solved. A clear example of a caveat not addressed yet is that hitherto mappings between a domain-specific language used for a CIM are not supported [235].

Maintenance - MDA also promotes the ease of conducting software maintenance. The mappings between the models are known, and thus the outcomes of the model-transformation are known and replicable. By making the mappings between models explicit tracing back the code to the original requirements is facilitated. [37, 118]. Rather than having to change individual modules or even bits of code, the models can directly be changed to make changes to the software.

While MDA might have strong merits, the discussion presented here highlights some important ramifications of using the MDA approach to develop software. These ramifications should be taken into consideration when designing systems based on the MDA philosophy. Informed by both the benefits of and considerations for MDA this study has set out to design the research approach.

4.4 Research Design

The benefits of employing the MDA led the writer of this dissertation to adopt it as the philosophy informing the design of the method to develop smart legal contracts that addresses the main research question. In this section the rationale to adopt MDA is explained and how it influenced the design of this research. An overview of the research method is provided thereafter.

4.4.1 Rationale for the Use of MDA

An important benefit of using MDA is that the use of models within our method supports the expertise of both legal experts and programmers. Figure 4.5 depicts the relation between the models and how they support various stakeholders during communication and carrying out their tasks. Because legal professionals are usually tasked with drafting legal contracts based on the requirements of the party they represent, the contracting parties themselves would not directly be involved in the creation of a smart legal contract.

Secondly, employing MDA to develop smart legal contracts promotes the communication between stakeholders by providing a lingua franca. This benefit is attained by the creation of a CIM that allows legal professionals to stipulate the requirements for their legal contract by modeling it using concepts they are familiar with. The creation of a CIM therefore addresses requirement 1. Another purpose of the CIM is to make potential issues when drafting and coding smart legal contracts explicit, in line with requirement 2 to aid programmers in addressing them. The result of the modeling efforts made by the legal professionals can be analyzed by programmers to specify a PIM for the smart legal contract, or alternatively let the method generate it. Because the relations between the concepts in the legal contract are known, problems of letting the smart legal contract enforce the legal contract can be foreseen which supports addressing requirement 3. Thereafter programmers can use the PIMs

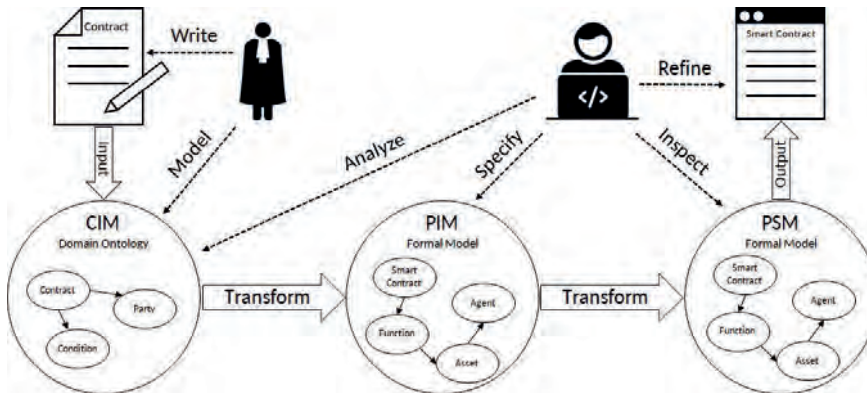


Figure 4.5: Research method in relation to stakeholders and requirements

to specify high level smart contract concepts that are to be translated into smart contract code.

Besides enhancing the ease of communication between the stakeholders, MDA allows for portability of specifications between platforms by the use of PIMs. The specification for the smart legal contract modelled as a PIM can be directly translated into the desired PSM. This feature greatly decreases the effort of writing smart legal contract for different platforms and thus increases productivity. More important is that smart contract related expertise required to create the models is only required once, and commonalities between the different platforms is reused. The method therefore supports requirement 5 and addresses the need for platform agnostic methods to create smart legal contracts. Bridges also enable the interoperability of specifications from one platform to another, further strengthening the potential of platform agnostic smart contract design.

MDA has also been reported to ease the maintenance of software once deployed. Although this benefit does not directly resonate with any of the requirements stipulated for the design of the method, the traces that are generated during the model transformation process for maintenance are. These traces are used to register the transformation of concepts between models. These traces would allow parties to generate smart legal contracts while being informed about how concepts are transformed. The information that the traces created during the transformation provide therefore coincide with requirement 4 that has been defined to diminish the reliance of the parties on third parties to translate their code.

4.4.2 Research Overview

Given this rationale we deem MDA an appropriate philosophy underpinning our method to develop smart legal contracts. The overarching method to develop smart legal is the main artefact that is designed to address the main research question. However, to realize the benefits of MDA a prescribed philosophy must be followed. The research design of this study was therefore designed in a manner that follows this philosophy. Figure 4.6 portrays the overall research design that has been employed to attain the results for this dissertation.

Employing the MDA philosophy requires the creation of a CIM, PIM and PSM(s), to enable software development. Each of these individual models is a distinct artefact constitutes to the overarching artefact requiring an idiosyncratic method to develop it. The distinct methods employed to develop each model are discussed in more detail in the chapters that

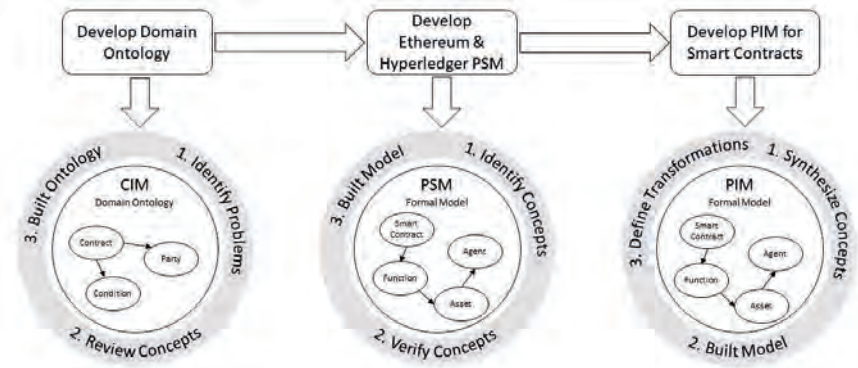


Figure 4.6: Overview overall research approach

present these models (Chapter 5 and Chapter 6). The first step to attain the results for this study was the development of a CIM. A CIM is a generic model that describes the system from a computation independent viewpoint. As discussed, they are described with a wide variety of modeling languages and manners. For this research we opted to design a domain ontology specified in the UML because they are specifically meant to capture the concepts in a domain. To construct the domain ontology, we first identified all issues related to developing smart legal contracts from the extant literature. Following, we identified the concepts used in a legal contract again through a literature review. The concepts and issues identified were used to inform the design of the domain ontology. By combining the insights attained about the issues and concepts, the relations between the both were laid bare. Finally, the domain ontology was evaluated by creating an instance based on a motivating example.

The domain ontology serves as the basis to capture the concepts used in the legal domain. When using MDA to generate code, it is common that thereafter these concepts are translated into one or more platform agnostic models. A metamodel to generate platform agnostic models for a smart contract is currently absent in literature. However, a PIM aggregates platform specific concepts that are shared across platforms. Thus, the concepts for the platforms targeted for the transformation are required to be identified first. Platform specific concepts were identified as second step to develop two PSMs for Hyperledger Fabric and Ethereum smart contracts. These two platforms are the most popular for smart contract deployment. A thorough examination laid bare the commonalities and differences between the concepts related to smart contracts that are used on these platforms. Informed by these two PSMs and the examination as a third step we set out to develop a platform agnostic model that can be used for multiple platforms.

4.5 Discussion

The use of MDA as a method to develop smart legal contracts introduces several advantages. Among these advantages are increased productivity, portability of code, interoperability between platforms, maintenance of the code, and the models themselves foster communication between stakeholders. While the MDA philosophy offers these advantages, like any approach it also has some drawbacks for smart legal contract development.

The creation of a CIM more specifically in the form of a domain ontology will aid in the communication between stakeholders. A CIM portrays all relevant concepts that are native to

a business and thus specify the domain itself. Although domain-specific languages are similar in that they also encompass domain concepts, the use of these languages still requires some programming skills that the legal professional in question might not possess. Models can remedy this problem by allowing the modelling of concepts instead of having to program these. Both domain-specific languages and domain specific models have an inherent drawback in that they only capture a limited set of concepts and thus are a simplified representation of that domain. Therefore, a domain-specific language or domain model cannot capture all of the nuances and particularities of a domain.

Another important advantage of MDA is that a PIM depicts an abstract representation of a system that can be used across different platforms. This is one of the most prominent advantages MDA has over traditional software development that requires developing distinct programs for each platform. However, at the moment much remains unclear on what techniques to use to transform a CIM into a PIM (see Section 4.3.3) making it hard to achieve a fully automatic transformation from the former to the latter. Some work has been conducted with regard to this problem for instance [87]. This might not be surprising however as domain ontologies or models are specific to that domain and therefore require an idiosyncratic set of transformation rules. Besides the aforementioned problem, by creating a PIM specialists run the risk of abstracting their model to the degree that much further specification is required when transforming the PIM to a PSM.

MDA offers bridges between platform software to enable a cross platform transformation. How these bridges are implemented largely determines whether this feature is possible however, and not all tools currently available support such a transformation [37].

The maintenance of programs built using the MDA philosophy is eased because mappings are known and traces are logged about how the transformation was executed. Perhaps the most interesting prospect of employing MDA to develop smart legal contracts is that the legal specification of the legal professional can be traced back to the code. This feature enables traceability between a contract and the smart contract. Consequently, this removes the need for yet another third party that transforms the legal specification to a smart contract. We argue that this is in line with the philosophy of BCT to remove third parties from the transaction process. Without explicit mappings between a CIM and a PIM this still difficult however.

Productivity can be increased by creating models that are re-used for multiple smart contracts. In addition, at the moment writing smart contracts is a labor-intensive process that demands vast amounts of expertise. By modelling smart contracts this expertise is captured and can be re-used for many developers. Domain experts that are to create the metamodels need to be well-educated about the inner workings of blockchain based smart contracts. Moreover, creating the models themselves however will require additional expertise because the designer of the model needs to be knowledgeable about UML.

4.6 Conclusion

To conclude, this research is conducted through the lens of the DSR paradigm. This paradigm is considered appropriate for this research as the direct aim of the study is to develop an artefact suitable to address a problem in a specific context. The complexity of the problem, combined with the required flexibility to develop a solution further confirms DSR as a suitable candidate to design the artefact. DSR itself does not specify any method to design or evaluate an artefact.

This research employs the MDA philosophy to fulfill the requirements stipulated for the method that this research aims to develop. MDA is a software development philosophy that employs models to ease the translation from domain specific knowledge to code. The chapter discusses the modeling levels in 4.3.1 used for MDA and relates these to the specific purposes they serve. A CIM aims to capture the domain for which the software is developed, including concepts related to that domain. Platform independent models portray the abstract and platform agnostic perspectives on a system. A further specification of the system for a particular platform is provided by a PSM. With MDA specifying models requires a metalanguage supported by an architecture. In 4.3.2 we present and discuss this architecture and how it is related to domain-specific languages and general-purpose modeling languages such as UML. At the heart of MDA is model transformation which is the process of transforming a PIM to a PSM. The role of model transformations and the mechanisms required to enable the process are discussed in Section 4.3.3. Mappings enable the transformation process, that can be divided into several categories and types.

In this chapter it has been argued why the advantages of MDA strongly resonate with the requirements stipulated for the artefact. From a technical perspective the approach support portability and platform agnostic abstraction while from a legal perspective it provides a basis for the specification of domain concepts and related requirements. To enable the use of MDA however, a CIM, PIM and PSM need to be developed. Hence, this research sets out to develop said models. In the next two chapters, Chapter 5 we will present the CIM in the form of a domain ontology, two PSMs and a PIMs for smart contracts in Chapter 6.

Chapter 5

Domain Ontology For Smart Legal Contracts

5.1 Introduction

As the digitization of society is becoming ever more pervasive in all aspects of life, the desire to express contracts and other legal agreements in a digital manner has gained momentum. The recently revived concept of smart contracts is a potential candidate for this application. Smart contracts are programming scripts deployed on a blockchain stipulating variables, conditions and functions that combined constitute to transaction logic [62]. However, currently smart contracts are application agnostic and not geared towards executing and enforcing legal agreements. The term smart legal contracts has been coined to describe smart contracts that digitally resemble a traditional contract.

With some exceptions [58, 82, 142, 150], to date most studies have considered smart contracts as a method solely to develop DApp's. Other Works on smart legal contracts [110, 273, 286] assume that the technical smart contract code *is* the contract.

This approach raises several problems however, the first being that contracting parties are required by law to understand the contract in most, if not all countries [64, 76, 80, 90, 179]. Not all legal practitioners are able to read code thus such an approach would not meet this requirement. Second, legal contracts have a complex structure. A direct representation of a contract as a smart contract is not be able to capture this complexity or the subsequent problems that arise from it. Third, while such approaches aim to represent concepts used in legal contracts directly as smart contract code it remains unclear if the set of concepts underpinning the model is complete and whether they are native to legal professionals.

An analysis of the concepts used in legal contracts is pivotal to attain an understanding of what concepts are relevant when modeling smart legal contracts. Domain ontologies are commonly used for this purpose. A domain ontology captures the concepts, and represents knowledge within a domain in a manner that is understandable by practitioners. This aids in creating a shared understanding of the information structure, promotes the reuse and analysis of domain specific knowledge, while making domain assumptions explicit [191].

This work presents a domain ontology to model smart legal contracts, and to delineate the relation among the elements contained therein. The ontology is specified in the Unified Modeling Language (UML) [220] to provide practical usability, and to cater for the transformation of the ontology into a blockchain related model like would be possible for a Computation Independent Model. An initial model was created by identifying concepts used to denote legal contracts from studies in the field of multi-agent systems and electronic contracts. Departing from prior studies, in the current work problems identified by other research are related to concepts used in legal contracts. The domain ontology presented here offers a notation for these problems so they may be incorporated in the notation of a contract.

The remainder of this chapter will be organized as follows: The section hereafter (Section 5.2), presents an overview of related works. In (Section 5.3) an overview will be provided of literature in the field of legal knowledge modeling, multi-agent systems and electronic contracts. Thereafter, in Section 5.4 the methodology followed to attain the results of this chapter will be discussed. Continuing, Section 5.5 will expound on the problems of drafting smart legal contracts and how they are related to the legal domain. To illustrate our ontology, in Section 5.6 a motivating example is presented that contains a condensed version of a contract. The 5.7th section presents the proposed domain model, and discusses the concepts described therein. Following, in Section 5.8 the domain ontology is evaluated by instantiating the motivating example presented in Section 5.6. In the discussion Section (Section 5.9) the results presented in this chapter will be further discussed. Finally, Section 5.10 concludes the Chapter.

5.2 Related Work: Modeling Smart Legal Contracts

Ladleif and Weske [150] suggest a unified model for legal smart contracts. Their model is based upon prior modeling languages and legal ontologies. As discussed, some researchers from the requirements engineering field have espoused their doubts whether these concepts are native to legal practice. Furthermore, the ontology only contains abstract concepts that are unsuitable for practical notation. Moreover, the ontology takes on a technical perspective that encompasses little legal concepts. To illustrate this point: it is commonplace in research on e-contracts like [10, 50, 127, 139, 184] to introduce the notion of clauses, yet this concept is missing in the work of Ladleif et al. In two subsequent works [149, 152] Ladleif suggest approaches to cater for the modeling and enforcement of smart legal contracts using BPMN. Boella et al. [34] argue however, that BPMN is unsuitable to model legal requirements as this would provide a oversimplified view of laws or legal relations.

Kruijff and Weigand [142] introduce an ontology for commitments based smart contracts. However, in their work concepts related to deontic logic are missing. These concepts are a crucial when stipulating the relation between legal clauses [57]. Furthermore, while the notion of definition clauses is introduced little guidance is provided as to how this concept is implemented. Again, this work takes on a technical angle towards developing smart legal contracts. Some other works [110, 273] propose to remedy this problem by introducing domain specific languages that are more alike legal prose and use nomenclature similar to that used by legal professionals. Despite that these efforts have greatly enhancing the legal realism and understandability of smart legal contracts, domain languages still require coding.

A recent review (May 2021) conducted by Dwivedi et al. [75] provides an overview of all current efforts regarding legally enforceable smart contract languages. In their review the authors suggest 10 critical properties categorized as semantic suitability, workflow suitability and expressive, that are required to constitute to a legally enforceable smart contract. An analysis reveals that none current works satisfies their suggested suitability and expressiveness properties.

Similar to this research other scholars have recently further examined the problems related to using smart contracts as legal contracts. In their very recent study Drummer and Neumann [74] analyze the challenges underlying the slow adoption of smart contracts in the legal domain. They dissect the shortcomings of smart contract into three levels: (1) How smart contracts conflict with current legislation, (2) the limitations of smart contracts on the contract level, (3) impediments caused by their current technical design. Although we identify several of the same problems as [74] there are some differences that sets this work apart from theirs. Firstly, we relate these problems to different phases of a smart contract life-cycle

whereas their work does not. Secondly, in this study other problems that we identify through an analysis of the literature are related to legal concepts like clauses or actions. Thirdly, the work by Drummer and Neumann also presents solutions for some of the problems they identified. However, the relation between these problems and legal concepts within a contract is not explicit, it is difficult to grapple the overarching relations on a contract level.

5.3 Digitizing Legal Contracts

The direction toward digitization of everyday processes has also taken effect on legal practice and contracting. Digitization refers to the process of converting information into a computable format (i.e. machine readable). Nowadays, there are multiple desperate fields like legal requirements engineering that aims to retrieve information legal knowledge from laws and contracts, multi agent systems literature seeking to model the interaction between contracting agents, while research on electronic contracts focuses on methods to model contracts and generate executable code. Initially however, most research endeavors were geared towards the digitization of laws.

The desire to digitize the logic of laws can be traced back to the writings of Gottfried Leibniz in 1664 that sought to reduce laws to mathematical computations. Already in his early writings Leibniz advocated the need for law to engage in an interdisciplinary dialogue, especially with logic and philosophy [11]. Over the past decades considerable efforts have been made to advance techniques that cater the digitization of laws. Legal requirements engineering is a field that focuses on extracting, modeling, and visualizing information denoted in laws or norms. The moniker norm encompasses a broad spectrum of "rules" that encompass laws and regulations, as well as social norms that are not necessary legally underpinned but are perceived as socially desirable. A common characteristic among norms is that they aim to govern the behavior of persons, institutions and organizations and prescribe what these entities ought to do, ought not to do, may or are allowed to do [271] in order to achieve a specific goal.

Studies in the field of legal requirements engineering have proposed several ontologies to model and conceptualize norms or laws. Two works by Breaux and Antón [40] and Massey et al. [174] present a methodology that caters for the analysis of rules with regard to privacy and security. Kralingen [136] suggested an ontology encompassing several frames that can be used to structure norms, acts, and the concepts stipulated in a law. The norm frame is used to capture information regarding the subject, modality and conditions under which the norm is applicable. Norms include several acts that in their own frame include temporal aspects, agents involved, and the aim of the act among things. The concept frame is used to denote definitions, deeming provisions, factors, and meta concepts. A study by Sleimi et al. [237] presents an ontology to conceptualize norms. Their work is based upon a review of ontologies used for legal knowledge modeling (UFO-L, NOMOS2, LKIF, GiausT). However, in a critical analysis Boella et al. [34] criticize the UFO-L, NOMOS2, LKIF, GiausT ontologies for lacking grounding in practice. In fact, they observe that most methodologies suggested to gather legal requirements are seldom developed in cooperation with legal practitioners. Consequently, such approaches present a simplistic view of law using concepts that are strange to legal practice.

Research in the field of electronic contracts does not only aspire to model concepts used in legal contracts, but to provide approaches to translate executable code that can enact transactions. In the past two decades electronic contracts (e-contracts) have become commonplace method to cater digitized agreements between parties. E-contracts are often used

in an e-service or e-commerce setting. In the former setting services are provided by a service provider to a recipient, while in the latter setting physical goods, funds, and data exchange ownership. In both settings an IT platform is employed to facilitate the enactment of the agreement [267]. The utilization of domain languages, formal procedures, logic and the standardization contract classes to cater digital agreements is collectively referred to as electronic contracting [127]. Works in the field of electronic contracting have touched upon some, or all of these aspects.

Lee [155] was among the first to pioneer the concept of e-contracts, provide a notation, and propose a concomitant logic model. However, it was not until 1995 with the publication of two founding papers by Marjanovic and Milosevic [171] and Daskalopulu and Sergot [65] that research in the field of e-contracting really gained momentum. The notion of electronic contracting was further enhanced in 1997 when Nick Szabo [245] coined the term "smart contract" referring to the automation of legal contracts. Crucially however, at the time a platform to monitor the execution of a contract, and safely exchange any assets between parties was absent [83]. Later works on electronic contracting [10, 50, 139, 184] have acknowledged this problem and did not only focus on modeling or providing formal languages, but in addition proposed architectures that support the enactment, monitoring and management of electronic contracts.

However, the introduction of these architectures and coherent models have led to a disparity of standards on how to model and enforce e-contracts. And, in practice none of the platforms has reached mainstream adoption. Other works [98, 184] present models only suitable to be used for specific types of contracts. Perhaps the most pervasive issue plaguing current e-contract solutions is their inability to enforce and manage payments [258]. This problem stems from the fact that e-contract architectures are not payment platforms and thus cannot execute payment orders.

5.4 Research Methodology

In carrying out our research we utilized a Design Science Research (DSR) [260] approach in which the design of an artifact is central. Given that the aim of this study is to construct a domain ontology for smart legal contracts as an artefact, we deemed a DSR approach appropriate. A first step in DSR is creating an initial version of the artefact. In constructing a domain ontology it is important to identify what terms, or concepts are used in the domain [191]. Another aim of this research is to lay bare what problems arise when using smart contract as legal contracts and how these are related to concepts used in contract practice. Thus we aim to answer the following research questions:

RQ1: What are the concepts related to legal contracts?

RQ2: What problems have been identified that are related to using smart contracts as legal contracts?

RQ3: How can the concepts related to legal contracts be modelled?

The first research question evolves around identifying problems related to using smart contracts in legal practice. The focus of RQ2 are the concepts or terms used in contracts. RQ3 seeks to offer an insight as to how these contract concepts are related by creating a domain ontology, while coupling these to the problems identified through RQ2. In designing the domain ontology presented here we followed the steps suggested by Noy and McGuinness [191]:

1. Determine the domain and scope of the ontology.
2. Consider reusing existing ontologies.

3. Enumerate important terms in the ontology.
4. Define the classes and the class hierarchy following a bottom-up or top-down approach.
5. Define the properties of classes—slots.
6. Define the facets of the slots (slot value type).
7. Create instances.

As a first step the domain and scope of the ontology was determined. For this research we have limited the scope of our ontology to legal contracts.

5.4.1 Data Gathering

Noy and McGuinness [191] suggest that as a second step the developer of an ontology should consider reusing existing ontologies. Considering that there are several works in the fields of legal requirements engineering, MAS and e-contracts that present metamodels, ontologies, languages and frameworks that describe digitized forms of legal contracts we informed the construction of our ontology with these studies. Therefore these literature strands were reviewed for relevant concepts to built our ontology with. For this research, a total of 19 papers were reviewed to identify relevant concepts.

Several legal practitioners and scholars have suggested that employing smart contracts for contracting introduces several problems. Thus, the author of this dissertation has in addition reviewed works that discuss these problems. In this second review 29 works were analyzed.

5.4.2 Data Analysis

After selecting a set of papers that contained concepts relevant for building our ontology these were analyzed to aid in answering RQ1. The analysis also aided in conducting the third step proposed by Noy and McGuinness [191]: the enumeration of important terms. Another analysis was performed to identify problems related to using smart contracts as legal contracts. Both analyses were conducted using a qualitative content analysis method [137] that encompasses the following steps:

1. *Select unit of analysis* - A first step was to select the unit of analysis for both sets of papers. For the execution of the analyses the author of this dissertation selected the paragraph level as the unit of analysis.
2. *Open coding* - As a second step, the author of this dissertation read all papers that were selected to extract concepts from for our domain ontology. Codes were applied to paragraphs that contained a concept. Then, the author went examined the set of papers that discuss legal problems related to smart contracts. Again, codes were applied to portions of the text that directly described a problem.
3. *Revise codes* - After the initial open coding process the codes were revised to create more uniformity among the codes that described a similar concept or problem. This step aided in reducing the number of codes, and easing the process of creating categories. For instance, the codes "activity" and "actions" were thereafter re-coded as action.
4. *Create categories* - The final step included the creation of categories for the concepts and the problems. A part of the process was to bundle concept codes and problem codes into an overarching category. This led to categories like events or actions that were directly used as classes in our ontology. Different codes were thereafter used to inform the design of sub-choices (e.g. clause types). The problem codes were categorized in the same vein, however the categories were coupled to

The content analysis performed on the set of works that contained concepts related to legal contracts yielded interesting insight on what concepts are used to specify legal contracts. An interesting observation is that while most works propose meta models, or entity relation models they do so with a different degree of granularity. Some works only propose concepts without directly relating other concepts like definitions. A definition always defines something (a word) as something else (things), yet this explanation is missing. Whenever this occurred further explanation was sought in literature not included in the initial set.

The results of the analysis of the works that discuss problems related to smart legal contracts reveals that these difficulties are related to distinct parts of the smart contract life-cycle. Further analysis, reasoning and reexamining of categories, codes and similarities led to the coupling of the identified problems to concepts identified. For example, as we will further explain hereafter the lack of a clause in a smart contract that stipulates the jurisdiction for the contract is a problem.

5.4.3 Constructing the ontology

Results from the analysis were used to construct the ontology and to answer RQ3. Following a bottom-up approach classes were created and a hierarchy between these classes was established as the fourth step. After the creation of the classes for each class its properties were defined. Some properties of the classes were not only the result of the analysis, but were also included for practical purposes. For instance, the Party class is given an id attribute that provides a unique id. As a fifth step the facets of the slots were defined. Finally, the last step suggested by Noy and McGuinness [191] is the creation of instances of the ontology. This final step serves to verify that the classes and terminology are suitable and sufficient to model the domain. The author of this chapter created instances of the domain ontology for four different types of contracts. In Appendix C an example of one of these contracts is shown, other instances of the contracts are accessible online¹ Other direct examples of instances are presented in listings hereafter.

5.5 Smart Contracts: Legal Challenges

The results of the analysis performed on the literature that discusses the difficulties of using smart contracts as legal contracts are presented in this section. Some of these problems directly related to the technical impediments of a smart contract. Other problems are directly related to legal concepts that are used for legal contracts. Further analysis reveals that these difficulties are related to distinct parts of the smart contract life-cycle. Table 5.1 provides an overview of the challenges identified that have been related to the smart legal contract life-cycle phases.

¹<https://github.com/BJBut/ContractInstances>

Table 5.1: Overview of Legal Challenges Related to Smart Contracts

| Lifecycle Phase | Nr. | Problem: | Problem Description: |
|----------------------------------|-----|-------------------------|---|
| Drafting & Coding | 1 | Understandability | A contract needs to be understood by all contract parties. |
| | 2 | Proving existence | Proving the existence of an agreement without a written version is difficult. |
| | 3 | Vagueness | Legal contracts can contain non-computable quantities of time and resources. |
| | 4 | Modality | The complexity of action modality complicates the specification of legal contracts. |
| | 5 | Ambiguity | Legal contracts can contain declarations of promises with coordinating conjunctions. |
| | 6 | Illegal clause | If a clause contradicts a law that clause might become illegal. |
| | 7 | References | Text in clauses might refer to other clauses and laws not directly accessible. |
| Testing | 8 | Unintended transactions | Transactions cannot be reverted even if the parties both agree to do so. |
| | 9 | Responsibility bugs | Which party bears the responsibility for bugs in the smart contract code is unclear. |
| Deployment | 10 | Applicable law | A specification of the applicable law might be missing. |
| | 11 | Jurisdiction | The jurisdiction that applies to a smart contract is sometimes lacking. |
| | 12 | Amendments | Once deployed a smart contract cannot be amended, while this might be required. |
| Signing | 13 | Signature requirements | Whether both a digital and analogue contract need to be signed remains unknown. |
| | 14 | Alignment of data | External data is not accessible to a smart contract if not on-chain. |
| Execution & Monitoring | 15 | Digital representation | Real world assets needs to digital representation to monitor ownership. |
| | 16 | Freedom of choice | Permissioned actions disable automatic execution of transactions by a smart contract. |
| | 17 | Wide range of actions | Contracts contain a wide range of action types that lack a representation. |
| | 18 | Flexibility | Smart contracts automatically enforce sanctions while parties might want to refrain. |
| Dispute Resolution & Termination | 19 | Termination clause | Without a termination clause a the conditions for terminating a smart contract are unknown. |
| | 20 | Undoing contract | A court can order to undo a contract or the contracting can agree to do so, yet blockchain transactions are irreversible. |
| | 21 | Force de Majeure | Situations can occur that discharges contracting parties from their obligations. |

5.5.1 Drafting and Coding

Scholars [64, 90, 179] and practitioners [76, 80] have suggested that it seems unlikely that smart contract (code) can be used without a conventional legal contract equivalent written in natural language because the contract is required to be *understandable*. For any legal contract to have a legally binding effect party's must be able to understand the contract. However, most legal professionals cannot read code, and moreover it is likely that the contractual party's concerned also cannot. Furthermore, when only a smart contract version of the contract exists *proving existence* in court of such a contract might be difficult [80, 187].

Several studies in the field of requirements engineering have identified complications related to translating laws to formal specifications [200]. Among these problems are *vaguely* formulated quantities for resources and vague deadlines. For instance, the phrase "that a Party shall make sufficient payments" is arbitrary. An example of a vaguely defined deadline is the use of demanding that an action is performed within "reasonable time". These complex terms are not measurable and therefore cannot be processed by a computer. Albeit that there is undesirable unintentional vagueness in contracts, contract lawyers often intentionally include vaguely defined quantities and deadlines to create leeway for unforeseen context-dependent contingencies. Hart [107] refers to this intentional vagueness within a law as open texture. The point of this explicit vagueness is to let the contractual parties decide when the situation occurs what they deem reasonable or sufficient [57, 64, 92, 96, 106, 236, 238]. Therefore problem with these vague deadlines and quantities is that they cannot be specified a priori without defeating their own purpose.

Ambiguity in contracts or laws further complicates the translation of these text into specifications that can be processed by programs [57]. In a study Kamsties et al. [125] identify multiple causes of *ambiguity*. Actions are often denoted with the coordinating conjunctions "and" and "or". For example, a party will pay the rent and the fee for electricity. The coordinating conjunction "and" indicates that both the rent and the electricity fee needs to be paid, which fact entails two to separate actions. Contrary, the coordinating conjunction "or" indicates an exclusive choice. A study conducted by Kamsties et al. [125] regarding the translation of laws shows that attempts to simply remove these ambiguities simply result in wrong or incomplete specification.

A related problem is that contracts contain complex *modal verbs* (e.g. shall, must, and may) that dictate the intended deontic logic [57]. However, the connotation of a modal and its relation to an action is dependent on the context it is used in. The modals will and would for instance, can mean an obligation or a forthcoming event. Modals that signal the negation of an action further increase complexity. However, understanding these modals are

quintessential to discern obligatory from permissioned and prohibited actions, and thus understand contracts [57].

Another potential issue is that conventional contracts contain clauses with actions that contradict a law that prohibits these actions, making them illegal. As a consequence, the (parts of) clauses that are contradicting the law may be declared invalid. Some authors [76, 80, 156, 223, 249] have argued that when translating these clauses to code the smart contract inherently also becomes *illegal*. A remedy used in conventional contract practice is to refer in the clause to the relevant article of the law, stipulating that whatever the law dictates will be in effect. Incorporating *references* to laws in smart contracts might prove to be difficult because it would require external access to a coded version of the corresponding law. Alternatively, the codified version of these laws would be stored in the smart contract itself. However, this would significantly increase the size of the smart contract. Other references point to articles or clauses in the contract itself which may be easier to translate [57, 64, 106].

5.5.2 Testing

After the smart contract has been written a testing phase commences to establish that it will execute and enforce the clauses in a contract as stipulated. According to Magazzeni, McBurney, and Nash [167] the testing phase further underpins the need for a written contract: If the expected behaviour for the smart contract is not specified then verification is impossible. For smart contracts the testing phase is paramount as *unintended transactions* when made cannot be reverted. Indeed, the irreversibility of smart contract transactions has been pointed out as an issue [230]. Unintended bugs in the code might cause unintended transactions which cannot be overturned. In these situations the question arises who shall *responsibility for the errors* [106, 230].

5.5.3 Deployment

The compiled version of a smart contract can be deployed on the blockchain platform once it has been tested. Uploading a natural language version of the conventional contract will ensure that third parties and the parties in contract themselves are able to read the contract. A smart contract is deployed on an international decentralized network and therefore transcend boundaries and have no physical location. Determining the *applicable law* for a smart contract is impossible if the parties in contract do not provide this information [76, 199, 217, 230]. Consequently, the *jurisdiction* for the appropriate court to settle a dispute between the parties is also unknown [80, 217, 230, 238]. Once deployed on a blockchain, smart contracts are immutable, impeding any *amendments* to their code. In traditional legal practice however, amending a contract is a common solution to settle disputes or to align the contract with the requirements of the law [236].

5.5.4 Signing

Once the smart contract has been deployed it can be signed by both parties to confirm their agreement. Nowadays several countries world wide have accepted forms of electronic contracts as legally enforceable for as long as these are digitally signed. For example, several states in the United States have passed the Uniform Computer Information Transactions Act (ESIGN). In the European Union the eIDAS enables electronic contracting. It is likely that smart contracts are equally enforceable if they are signed by the contracting parties [106, 270]. There is little clarity however about signature requirements when a smart contract represents a co-existing conventional contract.

5.5.5 Monitoring and Execution

Smart contracts are touted for their potential to independently whether conditions are satisfied. Data to evaluate conditions needs to be stored on-chain as smart contracts reside in a self-contained system [94]. Data not stored on-chain is therefore not accessible. Yet, to establish whether a transfer of physical resources has taken place a smart contracts must be able to attain data from the analogue world. This makes the *alignment of data* available to a smart contract and that required to evaluate reality difficult [179, 217, 238, 270]. Oracles have been suggested as a potential solution for smart contracts to attain external data. However, external data originating from oracles is required to be requested or provided to the smart contract, hampering automatic execution of the smart contract [179].

Another key benefit of smart contracts is the automatic execution of transactions based on conditional logic. However, some clauses in contracts permit parties to perform an action or refrain from it, providing a *freedom of choice*. The decision of a party to exercise its permission must be known to a smart contract before it can execute said action. Consequently this greatly diminishes the potential of a smart contract to automatically execute a transaction [179]. Another property that is often attributed to smart contracts is their ability to self-enforce the agreement in case a party does not uphold its promise. The rigid self-enforcement of a smart contract, combined with its automatic execution of actions confines the *flexibility* of the parties in contract to refrain from punitive sanctions when conditions in the contract are violated. Refraining from punitive sanctions is desirable under some circumstances, for instance to preserve relational contracts [64, 76, 80, 156, 179, 236].

A blockchain was initially envisioned as a system to facilitate transactions between parties without a trusted intermediary. As a result of this fundamental design focus, the actions a smart contract is able to perform is limited to transactions. However, contracts contain a *wider range of action* types that are equally important to represent. What portion of the other action types stipulated in a conventional contract can actually be performed by a smart contract remains a question [270]. Promises to exchange the ownership of assets are a commonality in contracts. To exchange the ownership of an asset existing in the real world it needs a *digital representation* on the blockchain to signal the transaction. Without this digital representation a smart contract also does not dispose over the data to monitor whether conditions are met [106].

5.5.6 Dispute Resolution and Termination

Termination of a contract occurs for several reasons, one of them being that one of the parties intentionally may want to breach the contract. If the benefits outweigh the potential benefits of a breach this might be a viable strategy for a party [217]. However, as a result of a breach sanctions might be applicable that are to be executed before the contract is terminated. In other contracts a *termination clause* might be missing, thus lacking clarity on when it is allowed to terminate the contract by right [236].

During the lifespan of the smart contract disputes between the parties might arise. For instance because one of the parties has breached the contract. When dispute resolution is required a common practice is to resolve the conflict via arbitration or litigation [96]. The lack of a clause that stipulates how disputes are resolved is problematic because oftentimes the contracts need to be *undone* as if the acts performed under the contract were never performed. The problem is that transactions executed using a smart contract cannot be reversed [217, 236, 270]. Another outcome of the dispute resolution might be that the contract is terminated by court order or mutual consent [170].

Finally, there are some events or conditions under which it becomes virtually impossible to perform the obligations. These events and conditions can constitute to *Force de Majeure* where a party is discharged from its obligations as it can not reasonably expected that they are performed [249]. However, even in the face of force de majeure smart contracts automatically execute or enforce the logic they contain once triggered.

In sum, there are the requirements that smart contracts need to meet in order to become a legally binding agreement while there are also more granular challenges related to distinct phases within the life-cycle of a smart contract. The problems discussed here constitute to genuine challenges when utilizing smart contract as an alternative to traditional legal contracts.

5.6 Motivating Example

Using a motivational example we will illustrate and demonstrate the domain ontology presented hereafter. The example is a condensed version of standard lease agreement, of which the full version can be found in Appendix C.

Lease Agreement

1. This Lease is made on 19th of July, 2021 between Bob Book, (hereafter called the Landlord) and John Doe, (hereafter called the Tenant).
2. The term of this lease is: of 2 Years starting on 1 January, 2021 and ending on 1 January, 2023.
3. If the Landlord cannot give possession within 30 days after the starting date, the Tenant may cancel this Lease.
4. The rent of the Premises will be \$1000.
5. The Tenant will pay the rent, in advance, on the 1st day of each month.
6. The first payment of rent and any security deposit is due by 01 January, 2021 prior to moving in.
7. The Tenant must pay a late charge of \$500 for each payment that is more than 30 days late.
8. The Tenant will deposit the sum of \$2000 with the Landlord as security that the Tenant will comply with all the terms of this Lease.
9. If the Tenant complies with the terms of this Lease, the Landlord will return this deposit within 30 days after the end of the Lease.
10. The Landlord may use as much of the security deposit as necessary to pay for damages resulting from the Tenant's occupancy or, at Landlord's sole option and election, to pay for delinquent or unpaid rent and late charges.
11. The Landlord will pay for the following utilities: Garbage Removal, Gas, and Oil.
12. The Tenant will pay for the following utilities: Water, Sewer, and Electricity.
13. If the Tenant does not pay the rent within 60 days of the date when it is due, the Tenant may be evicted.
14. The Landlord may also evict the Tenant if the Tenant does not comply with all of the terms of this Lease, or for any other causes allowed by law.
15. If evicted, the Tenant must continue to pay the rent during the remainder of the term.
16. The Tenant must also pay all costs, including reasonable attorney fees, related to the eviction and the collection of any monies owed to the Landlord, along with the cost of re-entering, re-renting, cleaning and repairing the Premises.
17. If the Premises are destroyed through no fault of the Tenant, the Tenant's employees or Tenant's visitors, then the Lease will end, and the Tenant will pay rent up to the date of destruction.

5.7 A Domain Ontology For Smart Legal Contracts

A contract is a legally binding written agreement between two or more parties, that have made mutual promises of performance towards one another [251, 283]. The actors in a contract are referred to as the contractual parties [50, 127, 283], that own assets pertaining a certain value. Definitions define the concepts used in a contract [40]. A contracting party has to perform a set of actions to full fill its promises to perform against the other party. From [10, 50, 127, 215, 283] we adopt the concept of clauses that specify the relation between a set of actions. The elements used to model contracts are clauses, definitions, parties, assets and actions. In Fig 5.1 the relation between these elements is depicted. The elements will now be discussed more in detail.

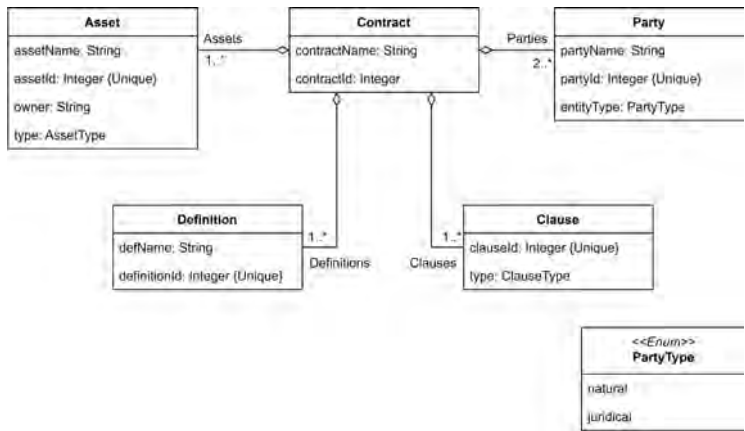


Figure 5.1: Composition of a contract

5.7.1 Asset and Party

Contract are always formed between two or more parties wishing to formalize their agreement in written form. Typically a contract starts with the recitals defining who the contractual parties are [127]. A definition could state who is a party involved in the contract, and more importantly how they are referred to in the contract. Example sentence 1 states for instance that "John Doe" is the Tenant, whereas "Bob Book" is the Landlord. Persons entering into a contract are considered parties to the contract. However, it must be noted that the term "person" actually refers to legal entities of which two types can be distinguished: First, there are natural persons, individuals like in our example. Second, there are juridical persons like companies. In our model we cater for this distinction by providing the field `entityType`, that can contain the value `natural` or `juridical`. The importance and ramifications of this design choice will be further explained in 5.7.4.2. Listing 5.1 portrays an example of how the party "Bob Book" mentioned in line 1 of the motivating example would be denoted.

```
1 {"party":
2   {"partyId": "Party1",
3     "partyName": "Bob Book",
4     "entityType": "natural"}
5 }
```

Listing 5.1: Example format Party

Parties own assets that are the subject of the promises made by the parties. For each asset the owner of an asset is registered in the field `owner` that can only contain the id of a party. Another important aspect of an asset is what kind of asset it is. Asset types like euro's are typically not unique, whereas a painting or a the lease of a house are. This distinction is required to determine the intrinsic value of the asset to a party. The `type` field is used to specify the type of asset.

5.7.2 Definition

Definitions represent concepts that are used throughout the contract and define that a particular concept is equal to another concept. As an example, "The rent" might be defined to be 5000 euro (See line 4 of motivating ex.). In essence, definitions pertain concepts that are relevant to understand the contract. Some definitions apply to most, if not all clauses stipulated in a contract. Again consider example line 1 stating who the Tenant and Landlord are. The composition of the elements related to a definition is depicted in 5.2.

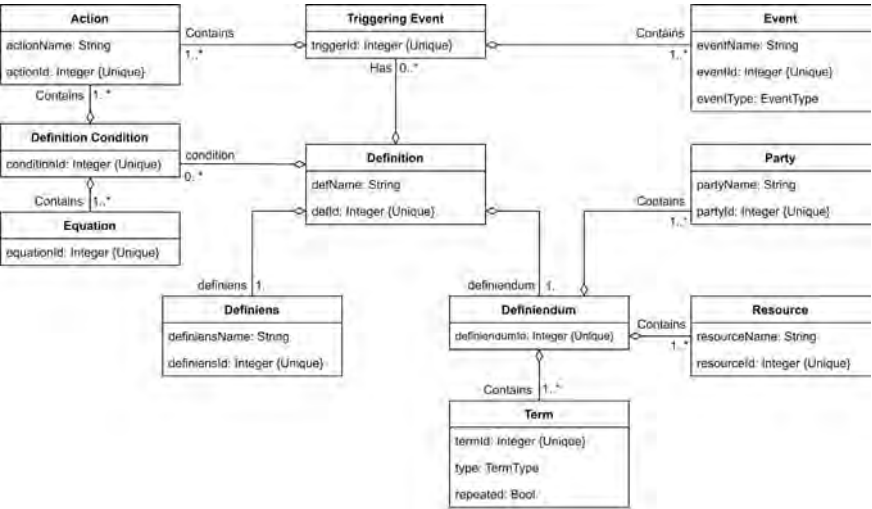


Figure 5.2: Model elements related to a definition

A Definition encompasses a definiendum and definens, a concept (definiendum) that is defined as another thing (definens). By expressing logical relations the limits of the definens is set. Such a logical expression could mean the equivalence of two things (A means B), or the inclusion of one more things (A includes B). In the specification When the inclusion of one or more things is defined, the definition will encompass several definens elements. Contrary logical expressions that include a negation, or a combination of the aforementioned logical expressions also exist. The quantities of the definiendum and definens are specified with either a `quantity` or an `arbitrary quantity`. Definitions can also only apply under a condition. For instance, a person can only be regarded as an adult for the purposes in a contract when that persons' age is over 18 (equation), or when something has occurred (action).

To illustrate how the domain ontology would cater for the notation of a definition, consider line 4 of the motivating example: "The rent of the Premises will be \$1000". How this line would be notated using the domain ontology is depicted in listing 5.2.

```

1  {"definition":{
2      "definitionId":"Definition4",
3      "definition":[
4          {"definiens":{"definiensName":"the rent",
5              "definiensId":"Definiens4"}},
6          {"defiendum":[
7              {"resource":{"resourceId":"Resource1",
8                  "resourceName":"$",
9                  "quantity":{"quantName":"the",
10                      "quantId":"Quantity1",
11                      "value": "500"}}}]]}]}
12 }

```

Listing 5.2: Example of modeled definition

In line with the domain ontology, and for identification purposes the definition has the id "Definition4". The most important information here is that "the rent" (see definiensName) is actually equal the amount of \$500. Therefore, in the definiendum a resource is enclosed since the definition actually defines a resource (Dollar) with a quantity value of 500.

5.7.3 Clause

A clause contains a subset of the promises to perform made by the parties in the form of several actions. In other works [50, 127, 139] on electronic contracting the structure of a clause is denoted in Event Condition and Action rules [19]. ECA rules take the form of On (event), If (condition), Do (actions) rules, meaning that on the occurrence of an event, if certain conditions are met a party executes an action. The structure of ECA rules resembles the notion of conditional promises that are promises made by a party to perform (an action) provided that the other party full fills its promise to perform an action. Related to the non-performance of a conditional promise, remedies might be stipulated in a contract. This work enhances the standard ECA rules by adding the notion of remedies, that are a specific type of actions only applicable when the conditions stipulated in the clause are violated. Figure 5.3 depicts the composition of the elements within a clause.

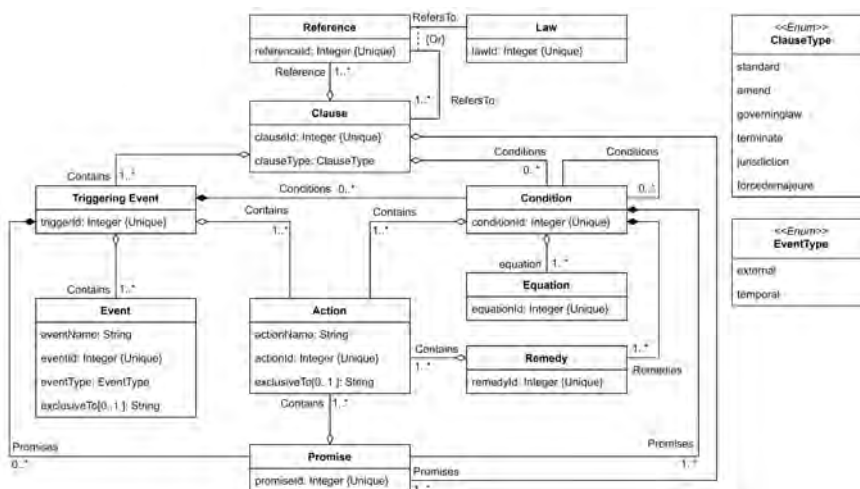


Figure 5.3: Model elements related to a clause

5.7.3.1 Triggering Event

Parties involved in the contract might perform actions, or an event might occur that prompts other actions to be performed by the parties. For instance, in line 8 the Tenants will make a deposit and the Landlord will return the deposit if the Tenant has complied with the terms of the lease. Hence, a contract event contains a set of actions that are the starting point for a clause to become applicable given a situation. We refer to the type of triggering events that are instigated by one of the parties in the contract as actions.

Not only a set of actions can effectuate a clause, a set of events can also be considered to make a clause applicable. We regard events as things that occur and which are not caused by the doing of one of the contractual parties. We discern two types of events (denoted as `eventType`): First, *external events* are occurrences of a change in the situation of the contracting parties that makes the performance of promises by another party pertinent. An external event or a set of external events that reasonably renders it impossible for a party to perform its promises is called Force de Majeure (See problem 21). The result of a Force de Majeure situation is that a Party may be promised to be discharged of its obligations. Second, *temporal events* are a are dates in the future that effect a clause.

5.7.3.2 Conditional Promise

A conditional promise is an action or event, that affects an obligation to render a promised performance of an action that is specified in a contract. Thus, a condition may be regarded as a qualification placed upon a promise [12], and the result of full filling that promise. Conditional promises stipulated in a clause may encompass one or more actions that are to be performed by the parties, or an evaluation of a situation based on certain criteria. The evaluation of a situation can be modeled as equations, that can be either true or false in the proposed model. Equations need to be evaluated first to determine whether the equation is true or false. In most contracts this entails evaluating the quantity of a resource to the quantity of another equated resource. A relational expression is used to denote how the resource and equated resource are evaluated. These relational expressions are greater than, equal to, smaller than, and equal to. The relations between the resources, equated resources and equated resources is further specified in Figure 5.4.

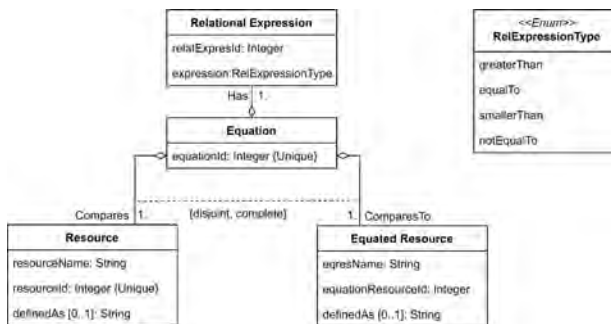


Figure 5.4: Model elements related to an equations

Conditional promises stipulated in a clause are always related to a promise to enact one or more actions stated in the clause. That is, when modeling legal contracts a clause cannot logically contain conditions without a promise to enact a set of actions. Clauses can contain multiple nested conditions where a condition entails the performance of another action,

that in turn full-fills a promise and renders the performance of a another promise. Besides conditional promises, a contract may contain unconditional promises. The performance of unconditional promises are as the name suggest, not tied to any condition.

5.7.3.3 Remedy

A clause might also contain remedies for whenever one of the contractual parties violates the conditions of the clause. A violation of a condition occurs when one of the contractual parties does not perform one or more actions that it has obligated itself to. When the violation of a condition occurs, the contract might describe remedies to resolve the breach, or a sanction. To illustration this: line 5 of the motivating example states that the Tenant promises to pay the rent. As a sanction line 7 states that if the the Tenant does not pay the rent, the Tenant must pay a late charge. Thus, a clause remedy contains an alternative set of actions that are enacted only if one of the contractual parties has failed to perform the actions specified in the clause conditions. From this line of reasoning it follows that a clause cannot contain remedies if there are no conditions formulated for the clause.

5.7.3.4 References

References denote the pointers to other paragraphs or sections in legal contract (e.g. in section 2 or in Schedule 1) [237]. In turn, these section can contain definitions or other clauses, effectively suggesting to read the referenced text to determine the outcome of a clause, or whether it is applicable. For each reference there is at least one corresponding clause or law to which it refers denoted in the field `refersTo`. References have been included in our model to address Problem 7 and cater resolution of such instances.

5.7.3.5 Combined Actions

Clauses might stipulate that a combination of actions is required to be performed to make itself applicable. In the same vein, such requirements might also hold for conditions and promises. Take line 6 of the motivating example, where the paying the "first payment" and "security deposit" are meant as a condition before being allowed to move in. Our model caters for combined actions by including these actions in one condition to denote that they are promised to be performed both. Whenever the combination of events with actions is required the same logic holds. Conversely, clauses triggers might state that the performance of an action is mutually exclusive to that of another. The coordinating conjunction "or" for instance, usually signals such exclusions. Whenever an exclusion is applicable the field `exclusiveTo` is used to capture the id's of actions, events, or a combination thereof respectively. Note that as exclusions are not always required, the `exclusiveTo` field is optional.

5.7.3.6 Clause Type

While other works [50, 127] consider clauses as one uniform element of a contract, or discern two types of clauses such as norm clauses and definition clauses [142, 182] we discern several types of clauses. A distinction between clauses is made by assigning a type for each clause. Examples of clause types are termination clauses, such as the one in line 3 or a clause stating the governing law. The lack of a termination, jurisdiction an applicable law clause is deemed problematic (See problem 10 and 11). We posit therefore that to model a smart legal contracts, each contract model needs to contain at least one of the aforementioned clause types. In the absence of a termination clause, the contractual parties may resort to

a standardized ("boilerplate") clause for contract termination. Similar, when the jurisdiction or applicable law are not explicitly specified, parties can choose to adopt a governing law or jurisdiction clause. In the same vein the absence of a clause to amend the contract introduces several problems (See problem 12). When modelling a contract there needs to be an amendment clause a boilerplate amendment clause may be when this clause is absent. Furthermore, the lack of a dispute resolution clause or mechanism is considered a problem within the current legal framework (See problem 20) as else contracts cannot be undone. Therefore, a smart legal contract model should include at least one dispute resolution clause. Finally, some but not all contracts specify the conditions that constitute to Force de Majeure (See problem 21). We suggest that a Force de Majeure clause is not required, yet if present should be taken into account.

5.7.4 Action

Conditions in a clause an subsequent promise of performance need to be materialized by parties through actions. For instance in line 5 of the motivating example the Tenant has committed to "pay" the rent. It can be noted that the verb *will* indicates that the action pay in the example is obligated. Furthermore, in the example the action has its own *scope* that comprises a resource (the Rent) and an agent (the Tenant). In other words, the parties that the action applies to, and the resources that are the target of the action. Finally, the action "pay" is to be performed by the Tenant, on the first day of each month meaning that temporal aspects are relevant. Furthermore, "each month" (time unit) indicates that the action is repeated. For identification purposes each action will have an unique identifier (`actionId`) and name that is derived from the action and the resource involved in the action. Figure 5.5 depicts the relation between actions, agents, resources and terms. Hereunder, we will further explain the relations between these concepts.

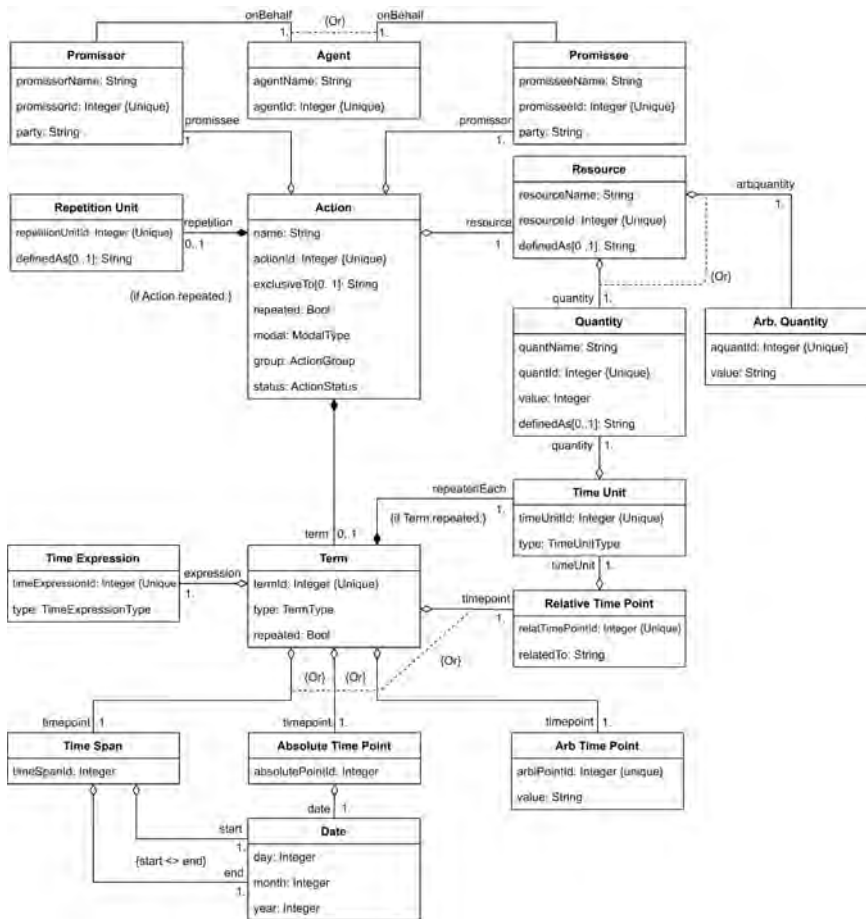


Figure 5.5: Relation between actions, resources, agents and terms.

5.7.4.1 Resources

In a contract there is usually an exchange of goods or services in other words, resources [31]. Examples of resources are a receipt, a lease deposit, 5000 euro's, or other materials. Example line 5 states only one resource. However, line 12 mentions that the tenant will pay for water sewer and electricity. While this seems one to be one action the statement actually comprises three actions: pay for water *and* for water *and* for electricity (See Problem 5). Thus, we can regard this as three separate actions that all need to be performed. The same line of reasoning can be followed for statements with an *or* coordinating conjunction. However, in the case of an coordinating "or" conjunction, the actions are exclusive and the same notation can be used as under 5.7.4.

Further required information is how much, or the quantity of the resource is involved in the action. In example 8, the quantity of the resource Dollar (\$) is clearly and directly stated as "2000" expressed as a number. The quantity of resources is not always directly stated but an article (e.g. "The", "A", or "Any") is used. Some articles can be expressed as real numbers, for instance "A" receipt can be said to mean 1 receipt. However, "The" suggests

that the specific quantity and resource are defined elsewhere. Take for instance line 5 of the motivating example that states "to pay the rent". The rent as a concept is defined in the contract to be \$1000 in line 4. To cater these links the notation presented here includes the field alias, that can be used to refer to the definition.

Another issue is that legal contracts are known to stipulate quantities of resources that are arbitrary expressions that are typically subjective like "reasonable" and "sufficient". A problem that arises from such statements is that they are not directly quantifiable (See problem 3). We refer to these as arbitrary quantities, that in our notation are mutually exclusive to quantities. That is, the quantity of any resource involved in an action is either arbitrary or in the form of a number.

5.7.4.2 Parties their Roles, and Agents

Each action in a contract needs to be performed by a party. When performing actions a party full-fills a specific role, that is related to whether it made the promise or not. For each action we discern two roles that a party may have, it is either:

1. A *promissor*, meaning that the party will perform the action, or:
2. A *promisee*, when the party is a recipient to an action.

A party that has the role of promissor will be tied to its role by denoting the corresponding identification number (`partyId`) in the field `party`. The same approach can be followed to denote the party that has a role as promisee. Ultimately the contracting parties themselves are responsible for full-filling their promises by performing actions.

However, in some contract scenarios a party is an juridical person (e.g. company), or like in the motivating example, a natural person. Whether the actions concern an juridical or natural person the actions need to be materialized by an individual. In line with other literature [31, 98], in our ontology we refer to these individuals as agents. It can be assumed that whenever a party is a natural party it will perform the promise itself. This might not be the case for a juridical entity like a company. Defining the legal relation between a party and an agent is complex and beyond the scope of this work. However, the field `onBehalf` is used to denote the link between a party and the agent operating on their behalf.

For some actions there is not always a receiving party stated, but only a providing agent. Line 5 of the motivating example illustrates this point. The contrary might also be true: contracts often contain such implicit statements regarding the providing and receiving agent of an action. Each action requires at least one agent that has the role of the providing agent and another agent that has the role of receiving agent. In the same vein as resources, when there is more than one agent related to an action this should be regarded as two separate actions.

5.7.4.3 Modality: Obligated, Permissioned and Prohibited Actions

The modality of actions are mostly described using deontic modal logic [271]. In literature regarding multi-agent systems [31, 234] and e-contracts [155, 171] three types of modality are discerned: an obligation, a permission and a prohibition. When an action is obligated it means that a party has promised to another party to perform an action. The Tenants promise in line 5 to pay the rent is an example of an obligation. Some actions are not obligated, but permitted meaning that they may be enacted. An example of a permissioned action is mentioned in first part of line 9: The Landlord may use as much of the security deposit. There are also prohibited actions that are not allowed to be performed by a party. In our notation, the modality of an action is denoted as an attribute of that action.

The debate about deontic logic has been influenced by the work of Hohfeld [113], that made significant efforts to identify fundamental legal concepts and establish the relationship between them. Hohfeld demonstrated that a duty (obligation) for a party, has as a natural opposite, a privilege (permission). A later work by Alexy [4] added the concept of duty-not (prohibition) which is the opposite from another parties right-not. From this reasoning it follows that actions cannot be obligated and permissioned simultaneously, nor can they be both prohibited and permissioned. Finally, actions can not be both prohibited and obligated as this would inevitably result in conflict [91]. Another important finding from the work of Hohfeld is that legal concepts like duty (obligation), right (permission), duty-not (prohibition) have correlative concepts. That is, the obligation for a party to perform an action is a liability, conversely it is the privilege of another party. In his work on norms Singh [234] maps normative concepts, that are akin to the aforementioned legal concepts, to the roles of a party. Based on this mapping we relate the proposed modalities to a parties' roles in Table 5.2.

Table 5.2: Liabilities and privileges by party role.

| | Promissor | Promissee |
|-------------|-----------|-----------|
| Obligation | Liability | Privilege |
| Permission | Privilege | Liability |
| Prohibition | Liability | Privilege |

5.7.4.4 Terms

Terms are deadlines that bind the performance of action to a point in future time. A term always contains a time expression, that indicates the relation between the time points. The complete set of types of time expressions that we discern is before, on, after, and between adopted from the work of Allen [6]. From other literature [155, 171] we identify three types of terms: absolute, relative, and timespan. Additionally we propose arbitrary terms. A time expression determines the type of the term, which we will now further explain.

A time expression might state that an action has to be performed *before* a specified date. Other time expressions require that an action was performed by one of the parties on a date, so that the performance of an action coincides with the date. Another form of a time expressions have a time expression stating that the action should be performed *after* a certain point in time. These are examples of *absolute* terms as the time point when the action needs to be finished is affixed. It can also be noted that absolute terms always specify a time point.

However, legal contracts might also contain *relative* term. An example of such a term is stated in line 9. The time expression "within" in this example is not related to a concrete date, but rather to time when the contract has been terminated. This example illustrates that actions that are bound to these types of terms are always preceded by one or more actions or the occurrence of an event. Rather than stipulating an absolute time point, relative terms have a relative time point. To determine the relative time point, the time point related to when an action was finished (`actionId`) or that an event has occurred (`eventId`) needs to be known. Further, the time unit stipulated in either days, weeks, months or years and the quantity thereof needs to be specified.

Other terms are arbitrary in that they have *arbitrary* time points that are subjective to interpretation. For instance the sentence "you will pay within reasonable time" is arbitrary as what is reasonable is subjective. Note that in this example a time point has not been specified. The time point needs to be agreed upon by the contractual agents involved in the

action. To facilitate this process the point needs an `arbiPointId` and a value that contains the expression (e.g. "reasonable" or "sufficient").

A term might be formulated in the form a *timespan* as it contains a time expression that the action is to be performed *between* two dates. Time spans always have a start date that specifies the start date of the time span, and an end date. Temporal constraints tied to an action can also be repeated, like in line 5 of the motivating example. On the 1st day of each month indicates a repeated payment for each time unit, in this example a month. If the term is repeated the repetition should be denoted in time units (e.g. months etc). Not all actions have an explicitly specified temporal constraint. In these cases it can be assumed that the action needs to be performed *eventually*, that is, before the contract is terminated.

5.7.4.5 Status of Actions

An important benefit of smart contracts besides the potential to automatically execute transactions is that a smart contract offers the possibilities to monitor the state of a contract. To achieve this goal, the state of each separate action needs to be captured. Transactions in a smart contract are atomic, therefore we assume that performance of an action is also atomic, they have either occurred or they have not. Here the input of the agent (performing an action) changes the status of an action from `unfinished` to `finished`. Given that an action still needs to be performed or has been performed by an agent the status for an action can have the value `finished` or `unfinished`. However, if there is a term tied to the performance of an action, and the action is not performed in line with this term, the conditions for that action are violated. We introduce a third status, namely that of a `violation` to denote this status. Therefore the status of all actions would be denoted as either `unfinished`, `finished` or `violated`. By default all actions are set to `unfinished`.

5.7.4.6 Action Group

A contract may stipulate a myriad of actions that need to be represented in a smart contract to cater for monitoring and execution. Actions in a contract could be transformed into specific code patterns that serve this purpose. It would be impossible to design and create a code pattern for each individual action. Even if this would be possible it would consume vast amounts of resources rendering this approach to be practically infeasible. Fortunately, when closely examining actions in contracts it can be noted that actions can be "grouped" based on their semantic equivalence. This observation is useful as it greatly reduces the number of patterns required to be designed.

To illustrate this point, the action "give" usually indicates that someone will provide something to someone else for instance. The action exchange has a similar meaning to the action give as it implies the change in ownership. Thus, these actions could be grouped into one group. Payments are related to the action "giving" yet specifically involves the exchange of currency. In the context of smart contracts this is a special group that requires a specific pattern. For other purposes assigning a group to an action might be equally beneficial. In section 5.7.3.6 we discussed the existence of several types of clauses. The types of actions that a clause contains can aid in identifying the clause type. Consider a sentence that includes the action "terminate", this action is usually mentioned in the context of a termination clause. Similarly, when the action "amend" is described this hints that the sentence is part of an amendment clause.

When we bring together all of this information and we take line 5 of the motivational example: "The Tenant will pay the rent, in advance, on the 1st day of each month." We can model the information concerning the action as shown in listing 5.3:

```

1  {"action": {
2    "actionId": "Action2",
3    "name": "pay the Rent",
4    "modal": "OBLIGATION",
5    "group": "Pay",
6    "repeated": false,
7    "status": "unfinished",
8    "resource": {
9      "resourceId": "Resource5",
10     "resourceName": "rent",
11     "definedAs": "Definition4",
12     "quantity": {
13       "quantName": "the",
14       "quantId": "Quantity5",
15       "value": 500,
16       "definedAs": "Definition4"
17     }
18   },
19   "promissor": {
20     "promissorName": "the tenant",
21     "promissorId": "Promissor2",
22     "party": "Party1"
23   },
24   "promisee": {
25     "promiseeName": "the landlord",
26     "promiseeId": "Promisee2",
27     "party": "Party?"
28   },
29   "term": {
30     "termId": "Term3",
31     "type": "span",
32     "timeexpression": {
33       "timeExpressionId": "TimeExpression2",
34       "type": "on"
35     },
36     "timePoint": {
37       "timeSpan": {
38         "timeSpanId": "TimeSpan2",
39         "start": {
40           "date": {
41             "day": 1,
42             "month": 1,
43             "year": 2021
44           }
45         },
46         "end": {
47           "date": {
48             "day": 1,
49             "month": 1,
50             "year": 2023
51           }
52         }
53       }
54     }
55   },
56   "repeatedEach": {
57     "timeUnitId": "TimeUnit1",
58     "type": "month",
59     "quantity": {
60       "quantName": "1",
61       "quantId": "Quantity6",
62       "value": 1}}}}

```

Listing 5.3: Example of modeled action

A first important observation from the example is that the resource involved in the action "paying" is "the rent". However, the rent is not a resource or a direct quantified amount. What the rent actually is is defined somewhere else, more specifically in line 4 of the motivational example. The example of how the ontology could be used to denote this definition is shown in listing 5.2. Therefore, in listing 5.3 the resource name is "rent" while the quantity

name (quantName) is set to "the". To cater for the reference to the correct definition, the field definedAs contains the name of the definition where the rent is defined. A second important observation is that in the example the action must be repeated each month. In listing 5.3 the attribute repeated is thus set to "true". The action has a timePoint with the attribute repeatedEach to stipulate that the action is repeated each month. What is also important to denote is the time span during which the action needs to be required. In this case the rent needs to be paid by the tenant from the start of the contract, 1st of January 2021 to the end of the contract on the 1st of January 2023. Both dates are denoted in the attribute timeSpan under start and end.

5.8 Evaluation: Instantiating the Domain Ontology

In line with the final step proposed by Noy and McGuinness we created instances of the ontology to evaluate whether it can be used in practice. When following Noy and McGuinness' methodology this final step serves to evaluate an ontology based on one or multiple cases. The motivating example presented in section 3.2 and has been used throughout this chapter to exemplify the concepts of the ontology has been instantiated, and is shown in Appendix D. Given that the modeling of individual concepts has been discussed in the prior sections we emphasize on explaining how clauses are modelled. For brevity's sake, we will illustrate the instance of only one clause of the motivating example here (shown in listing 5.4).

```

1  {"clause": {
2      "clauseId": 1,
3      "clauseType": "Normal",
4      "TriggeringEvent": {
5          "triggerId": "Trigger1",
6          "Contains": [
7              {
8                  "event": {
9                      "eventName": "start lease",
10                     "eventId": "Event1",
11                     "eventType": "temporal"
12                 }
13             }
14         ],
15         "Conditions": [
16             {
17                 "condition": {
18                     "conditionId": "Condition1",
19                     "Contains": [
20                         {
21                             "action": {
22                                 "actionId": "Action1",
23                                 "name": "pay the Rent",
24                                 "modal": "OBLIGATION",
25                                 "group": "Pay",
26                                 "repeated": false,
27                                 "status": "unfinished",
28                                 "resource": {
29                                     "resourceId": "Resource4",
30                                     "resourceName": "rent",
31                                     "definedAs": "Definition4",
32                                     "quantity": {
33                                         "quantName": "the",
34                                         "quantId": "Quantity4",
35                                         "value": 500,
36                                         "definedAs": "Definition4"
37                                     }
38                                 }
39                             },
40                             "promissor": {
41                                 "promissorName": "the tenant",
42                                 "promissorId": "Promissor1",
43                                 "party": "Party1"
44                             }
45                         }
46                     ]
47                 }
48             }
49         ]
50     }
51 }

```

```

42         "promissee": {
43             "promisseeName": "the landlord",
44             "promisseeId": "Promissee1",
45             "party": "Party2"
46         },
47         "term": {
48             "termId": "Term3",
49             "type": "span",
50             "timeexpression": {
51                 "timeExpressionId": "TimeExpression2",
52                 "type": "on"
53             },
54             "timePoint": {
55                 "timeSpan": {
56                     "timeSpanId": "TimeSpan2",
57                     "start": {
58                         "date": {
59                             "day": 1,
60                             "month": 1,
61                             "year": 2021
62                         }
63                     },
64                     "end": {
65                         "date": {
66                             "day": 1,
67                             "month": 1,
68                             "year": 2023
69                         }
70                     }
71                 }
72             }
73         },
74         "repeatedEach": {
75             "timeUnitId": "TimeUnit1",
76             "type": "month",
77             "quantity": {
78                 "quantName": "1",
79                 "quantId": "Quantity5",
80                 "value": 1
81             }
82         }
83     }
84 },
85 "Remedies": [
86     {
87         "remedy": {
88             "remedyId": "Remedy1",
89             "Contains": [
90                 {
91                     "action": {
92                         "actionId": "Action2",
93                         "name": "pay a late charge",
94                         "modal": "OBLIGATION",
95                         "group": "Pay",
96                         "repeated": false,
97                         "status": "unfinished",
98                         "resource": {
99                             "resourceId": "Resource5",
100                             "resourceName": "late charge",
101                             "definedAs": "Definition6",
102                             "quantity": {
103                                 "quantName": "the",
104                                 "quantId": "Quantity6",
105                                 "value": 500,
106                                 "definedAs": "Definition6"
107                             }
108                         }
109                     },
110                     "promissor": {
111                         "promissorName": "the tenant",
112                         "promissorId": "Promissor2",
113                         "party": "Party1"

```

```

114         },
115         "promissee": {
116             "promisseeName": "the landlord",
117             "promisseeId": "Promissee2",
118             "party": "Party2"
119         },
120         "term": {
121             "termId": "Term4",
122             "type": "relative",
123             "timeexpression": {
124                 "timeExpressionId": "TimeExpression4",
125                 "type": "after"
126             },
127             "timepoint": {
128                 "relativeTimePoint": {
129                     "relatTimePointId": "
130                         relatTimePoint1",
131                     "relatedTo": "Action1",
132                     "timeUnit": {
133                         "timeUnitId": "timeUnit2",
134                         "type": "day",
135                         "quantity": {
136                             "quantName": "30",
137                             "quantId": "Quantity7",
138                             "value": 30
139                         }
140                     }
141                 }
142             }
143         }
144     }
145 }
146 {
147     "remedy": {
148         "remedyId": "Remedy2",
149         "Contains": [
150             {
151                 "action": {
152                     "actionId": "Action3",
153                     "name": "evict tenant",
154                     "modal": "PERMISSION",
155                     "group": "terminate",
156                     "repeated": false,
157                     "status": "unfinished",
158                     "resource": {
159                         "resourceId": "Resource6",
160                         "resourceName": "evict",
161                         "quantity": {
162                             "quantName": "the",
163                             "quantId": "Quantity8",
164                             "value": 1
165                         }
166                     }
167                 },
168                 "promissor": {
169                     "promissorName": "the landlord",
170                     "promissorId": "Promissor3",
171                     "party": "Party2"
172                 },
173                 "promissee": {
174                     "promisseeName": "the tenant",
175                     "promisseeId": "Promissee3",
176                     "party": "Party1"
177                 },
178                 "term": {
179                     "termId": "Term5",
180                     "type": "relative",
181                     "timeexpression": {
182                         "timeExpressionId": "TimeExpression5",
183                         "type": "after"
184                     }
185                 },
186                 "timepoint": {
187                     "relativeTimePoint": {

```



```

185         "relatTimePointId": "
186             relatTimePoint2",
187         "relatedTo": "Action1",
188         "timeUnit": {
189             "timeUnitId": "timeUnit3",
190             "type": "day",
191             "quantity": {
192                 "quantName": "60",
193                 "quantId": "Quantity9",
194                 "value": 60
195             }
196         }
197     }
198 }
199 }
200 }
201 ]}}}}},
202 "Promises": [
203     {
204         "promise": {
205             "promiseId": "Promise1",
206             "Contains": [
207                 {
208                     "action": {
209                         "actionId": "Action4",
210                         "name": "pay the security deposit",
211                         "modal": "OBLIGATION",
212                         "group": "deposit",
213                         "repeated": false,
214                         "status": "unfinished",
215                         "resource": {
216                             "resourceId": "Resource7",
217                             "resourceName": "security deposit",
218                             "definedAs": "Definition5",
219                             "quantity": {
220                                 "quantName": "the",
221                                 "quantId": "Quantity10",
222                                 "value": 2000,
223                                 "definedAs": "Definition5"
224                             }
225                         },
226                     },
227                     "promissor": {
228                         "promissorName": "the tenant",
229                         "promissorId": "Promissor4",
230                         "party": "Party1"
231                     },
232                     "promissee": {
233                         "promisseeName": "the landlord",
234                         "promisseeId": "Promissee4",
235                         "party": "Party2"
236                     },
237                     "term": {
238                         "termId": "Term2",
239                         "type": "absolute",
240                         "timeexpression": {
241                             "timeExpressionId": "TimeExpression2",
242                             "type": "before"
243                         },
244                         "timePoint": {
245                             "absoluteTimePoint": {
246                                 "absolutePointId": "TimePoint1",
247                                 "date": {
248                                     "day": 1,
249                                     "month": 1,
250                                     "year": 2021
251                                 }
252                             }
253                         }
254                     }
255                 }

```

```

256     }},
257 {
258   "promise": {
259     "promiseId": "Promise2",
260     "Contains": [
261       {
262         "action": {
263           "actionId": "Action5",
264           "name": "pay for gas",
265           "modal": "OBLIGATION",
266           "group": "Pay",
267           "repeated": false,
268           "status": "unfinished",
269           "resource": {
270             "resourceId": "Resource8",
271             "resourceName": "gas",
272             "quantity": {
273               "quantName": "the",
274               "quantId": "Quantity?",
275               "value": "?"
276             }
277           },
278           "promissor": {
279             "promissorName": "the tenant",
280             "promissorId": "Promissor5",
281             "party": "Party1"
282           },
283           "promisee": {
284             "promiseeName": "the landlord",
285             "promiseeId": "Promisee5",
286             "party": "Party2"
287           },
288           "term": {
289             "termId": "Term6",
290             "type": "span",
291             "timeexpression": {
292               "timeExpressionId": "TimeExpression6",
293               "type": "during"
294             },
295             "timePoint": {
296               "timeSpan": {
297                 "timeSpanId": "TimeSpan3",
298                 "start": {
299                   "date": {
300                     "day": 1,
301                     "month": 1,
302                     "year": 2021
303                   }
304                 },
305                 "end": {
306                   "date": {
307                     "day": 1,
308                     "month": 1,
309                     "year": 2023
310                   }
311                 }
312             }
313           }
314         }
315       }
316     ],
317     {
318       "action": {
319         "actionId": "Action6",
320         "name": "pay for water",
321         "modal": "OBLIGATION",
322         "group": "Pay",
323         "repeated": false,
324         "status": "unfinished",
325         "resource": {
326           "resourceId": "Resource9",
327           "resourceName": "water",

```

```

328         "definedAs": "Definition5",
329         "quantity": {
330             "quantName": "the",
331             "quantId": "Quantity11",
332             "value": 2000
333         }
334     },
335     "promissor": {
336         "promissorName": "the tenant",
337         "promissorId": "Promissor5",
338         "party": "Party1"
339     },
340     "promissee": {
341         "promisseeName": "the landlord",
342         "promisseeId": "Promissee5",
343         "party": "Party2"
344     },
345     "term": {
346         "termId": "Term6",
347         "type": "span",
348         "timeexpression": {
349             "timeExpressionId": "TimeExpression6",
350             "type": "during"
351         },
352         "timePoint": {
353             "timeSpan": {
354                 "timeSpanId": "TimeSpan4",
355                 "start": {
356                     "date": {
357                         "day": 1,
358                         "month": 1,
359                         "year": 2021
360                     }
361                 }
362             },
363             "end": {
364                 "date": {
365                     "day": 1,
366                     "month": 1,
367                     "year": 2023
368                 }
369             }
370         }
371     }
372 }
373 ]
374 }
375 }
376 }
377 ]}]},

```

Listing 5.4: Instance of a clause within the motivational example

The first thing that needs to be established is which actions are part of a clause. In case of the motivating example the clause becomes applicable at the start of the contract (see line 2 motivating example) that is modelled in listing 5.4 as an a temporal event under the attribute `TriggeringEvent` (See line 5 to 12 in Listing). Attached to this event is the condition presented in line 5 of the motivational example. The modeling of the action has already been discussed prior and is shown in listing 5.3. Two specific remedies are mentioned when the tenants does not pay the rent and violates the condition. The first of these remedies is that the tenant shall pay a late charge of \$500 for each payment due after 30 days (line 7 motivational example). Thus, remedy 1 containing Action 2 on line 92 is included in the model for such circumstances. However, if any payment is missed then the landlord is allowed to evict the tenant after 60 days and effectively terminating the lease. Note that because this not an obligation but rather a permission, the modal on line 95 of Action 3 is set to permission. Similar to Action 2, in the information for Action 3 a relative timepoint is included with the

attribute `relatedTo` to signal that the related action here is Action 1. Note that the group attribute is set to terminate.

The motivational example also includes the promise of the Tenant to pay a security deposit (line 8 of motivational example). In the model this becomes promise 1 starting on line 205 listing 5.4. There are no conditions tied to the promise to pay the security deposit when the contract commences. Besides this promise, the Landlord promises to pay for Garbage Removal, Gas, and Oil utilities in line 11. The tenant will pay for the Water, Sewer, and Electricity utilities as described in line 12. All of these promises are described in lines 259 through 377. For brevity's sake we have not included all actions here, but paying for Garbage Removal, Gas, and Oil utilities should be regarded as separate actions. Similarly, so should paying for Water, Sewer, and Electricity utilities.

So far, we have presented how information about the contract is stored using the domain ontology. As the final usability test of the domain ontology, the information about the contract is implemented in Contract Custodian². Figure 5.6 depicts the motivational example when used in combination with Contract Custodian a green box indicates a frame that will now be explained.

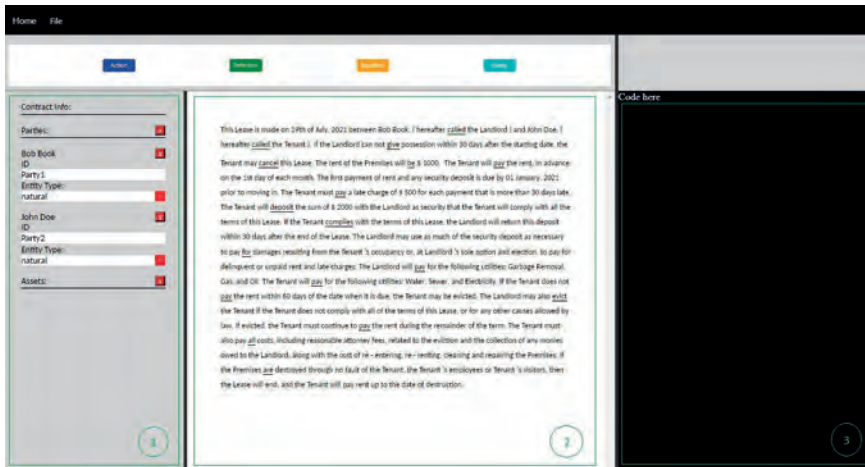


Figure 5.6: Contract level concepts displayed in Contract Custodian

The first frame shows contract level information such as the parties and assets. The contract information can be changed using several buttons. In the second frame the contract text is shown with all actions, definitions and events underlined in blue, green and turquoise respectively. The third frame is black and shows a field where could generated using the contract model is placed. Because the code is placed directly next to the contract text a direct comparison is possible thus facilitating direct traceability between text and code. End-users will also need require information at the action, definition or event level. To visualize this information, end-users can hover over actions, definitions and events. Basic information about these concepts can be shown directly in this manner. A more detailed overview is created once the end-user has clicked on the concept. Figure 5.7 shows Contract Custodian when a concept is clicked.

²Contract Custodian is a program written by the author of this dissertation to visualize the information in the contract. Please note that the program is a Proof of concept, and work in progress.

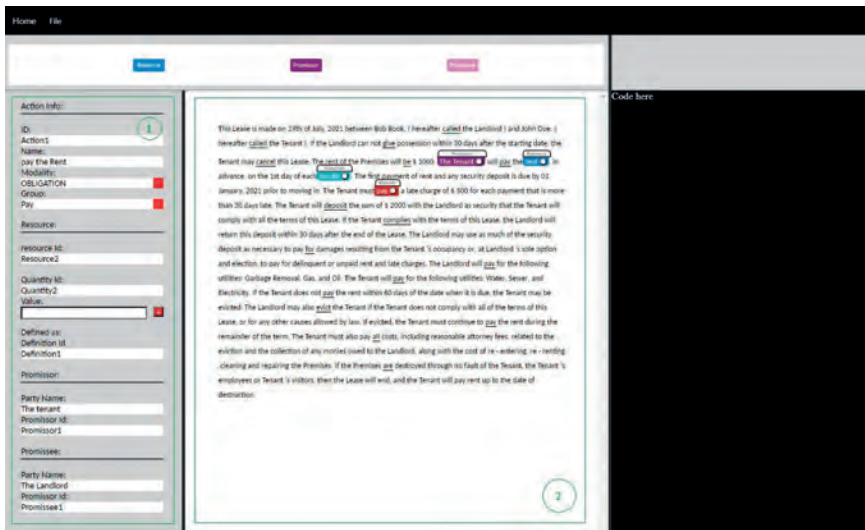


Figure 5.7: Action level concepts displayed in Contract Custodian

In frame 1 in Figure 5.7 the action level is again showed, along with the information that is in the contract text. All generic information about the action is shown in the top part of frame 1. Note that although in the sentence the promisee is not mentioned (the Landlord), it is shown in the menu as there is always a promisee for each action. Further, the resource involved in the action is shown. The second frame contains again the contract text. However, when an action is activated by clicking it information related to action is shown in the text. With several colors things like where the resources and information is mentioned is highlighted in the text. The relation that the action has with other actions is also displayed. In this case for instance, the action "pay a late charge" is shown as a remedy.

5.9 Discussion

The domain ontology presented in this chapter serves as a basis to model contracts as smart legal contracts, and to delineate the relation among elements contained therein. Concepts from related work was used to design the domain model.

An observation we make is that currently researchers from several desperate fields are making related efforts, with different accents however. Most studies on legal requirements engineering predominantly focus on capturing information with regard to norms, not on how the enforcement of these norms can be monitored. On the contrary, multi agent systems focus on the interactions between agents yet might lack the depth of an ontology to support contracts. The current division of efforts has led to a fractured research landscape. Researchers conducting work on e-contracts have attempted to propose solutions that include various aspects of electronic contracting. Despite the advances and the subsequent invaluable insights the field has offered, each solution has idiosyncratic platform supporting the e-contract. An implication of the diversification of these platform is that the exchange of data or value between platforms might be hampered. More important, parties engaging in electronic contracting have little choice than to trust the platform that they are utilizing in terms of security because the security standards might be unknown for the different platforms.

Smart legal contracts provide the unique opportunity to unite the efforts of several related fields under one banner. Their underpinning by a blockchain means that a trustworthy exchange of value between parties is possible. Other studies presenting ontologies for commitment based smart contracts [142] or smart legal contracts ([273] and [110, 286]) have taken on an approach where the code is the contract of smart legal contracts. A review of literature reveals that this approach will introduce several problems, not in the last place that a code is not understandable. In an attempt to resolve this problem Ladleif and Weske [149] present a BPMN based notation to stipulate the logic of the contract. Yet, some scholars [34] have argued against BPMN approaches to model laws and legal relations. In addition, we posit that a BPMN approach to model smart legal contracts might introduce several concepts that are not native to legal practice thus hampering the understandability of what is actually modeled.

Reviewing research on e-contracts reveals that clauses are among the most commonly used concepts. The inclusion of particular types of clauses in a contract (e.g. termination, jurisdiction) are required to avoid enforcement problems. Seeking to find a remedy for this problem, we suggested that clauses can be classified into types. In 5.7.3.6 four types of clauses are discussed that are derived from literature [50, 127]. Milosevic, Sadiq, and Orlowska [182] discuss definition, termination, and reporting and notification clauses, that are all included as types in the ontology except the reporting and notification clause. However, hitherto there is no all-encompassing overview of the types of clauses that are used to formulate a contract. Similarly, in Section 5.7.1 we introduce the concept of assets. A list of asset types is not included. While the discussion in the section hints at discerning several asset types based on their uniqueness, to our knowledge there exists no exhaustive classification list of asset types described in legal contracts. Some work has been done in this regard though. Building on the work of He et al. [110], discern several types of assets like digital currency assets, data assets, physical assets, intangible assets. However, it must be noted that it remains unclear how the authors established this set of asset types and whether it is exhaustive. Further research is needed to verify whether this fact holds true.

In our ontology clauses can encompass several conditional promises to perform. From a legal perspective conditional promises to perform are akin to concurrent conditions, that have to hold during the execution of the contract. In legal practice two additional types of conditions are also used [12]: conditions *precedent* and conditions *consequent*. A condition precedent refers to conditions which require that an event occurs before the promisor(s) are obligated to perform any of their promises become due. Line 3 of the motivating example presents a condition precedent. On the other end, conditions subsequent entail that a party will be discharged from its promises to perform under the contract whenever (1) some event occurs, or (2) fails to continue to occur. In line 15 of the motivating example such a condition is stipulated.

Remedies in our ontology serve as an alternative set of actions to a promise whenever a conditional promise to perform is violated. Some studies refer to actions as sanctions. Contrary to sanctions, remedies are not always intended to punish a person, while sanctions are always punitive. Nevertheless, remedies and sanctions are similar in that they refer to a loss that a party must bear if it is found liable when it does not uphold its promise [79]. Therefore no distinction has been made between sanctions and remedies in our ontology. Another aspect about remedies to consider is that some contracts stipulate specific remedies for a given situation while others do not. However, most legal traditions also provide supplementary remedies based on laws.

A problem left partially unaddressed by our ontology is that legal contracts can contain several types of actions like, pay, send or others. The ontology provides a manner to specify the

type of actions (See Section 5.7.4) based on a group. However, we do not provide a list of actions that users of the domain ontology can use to stipulate the type of action. In the future research endeavours could be committed to empirically investigate what types of actions are described in contracts.

The use of references in a contract has been signaled as problematic (See problem 7 in Section 5.5). References to clauses might be easy solvable as the related information is contained within the contract. The same might not hold true for references to laws as this information is stored outside of the contract. Solutions might be close however, recently an executive decree from the presidents office in the U.S. ordained that every law should become machine readable [192]. To date this remains an unresolved problem.

We have limited the roles that a party can have for promises to that of promisor and promisee. Although beyond the scope of this work, other role might be discerned. In U.S. contract law for instance, besides the roles of promisor and promisee a party can have the role of beneficiaries. Beneficiaries are third-parties that, despite not being an active party in the contract, attain certain rights from a contract. Our ontology caters for the fact that an individual agents materialize the promises to perform on behalf of a party, especially when the party is a judicial entity. However, other authors [234] include the idea of an authorization to act on behalf of this party or the concept of delegation [123]. The ontology presented here does not include a notation to attribute an agent with an authority. Neither is there a direct manner to verify whether an agent is actually authorized to perform the action on behalf of a party. This is a complex issue that needs further investigation that is beyond the scope of this work.

Overall the domain ontology presented here provides a starting point for discussion within the field of smart contracts, and we hope serve as a stimulus to engage in a dialogue with legal practitioners. A thorough discussion where practice and science meet will arguably contribute to an international standard for smart legal contracts.

5.10 Conclusion

The main goal of the research presented in this chapter was to construct a domain ontology for smart legal contracts that can be used to specify the design of a smart contract. By reviewing the extant literature on legal requirements engineering, multi-agent systems and electronic contracts we identify several concepts required to model smart legal contracts. Examples of these concepts are the notion of a contract, between party's that own assets. Clauses govern the behaviour of the party's and contain the (conditional) promises that the parties agreed upon to perform. Each promise must be performed by a party by undertaking an action. The relations among these concepts is also explained, further enhancing the current knowledge on how smart contracts can be used to represent legal contracts. More important it provides legal practitioners and with a lingua franca for the design and creation of smart legal contracts.

Heeding the warnings and critiques espoused by both scholars and practitioners however, this research commenced by first identifying problems related to using smart contracts to represent legal contracts. A total of 21 issues was identified. Departing from prior research we acknowledge these problems, and specify how the concepts in the ontology are related to these problems. With the aim of making these problems explicit we couple these directly to the concepts in the domain ontology. In doing so we make domain explicit and provide legal practitioners and software engineers with a common understanding about these problems.

The domain ontology presented in this work is not without limitations. We argue that several types of assets, clauses, action groups can be discerned. However, we do not provide an extant or exhaustive list for each of these concepts. Although we acknowledge this as a current limitation of the ontology presented here, we designed the ontology in a manner that can be extended to cater for different types. Besides these limitations the ontology currently lacks the possibility to specify that a certain agent is authorized to perform an action.

We hope that the domain ontology presented in this chapter will provide a starting point for further dialogue between the legal community and the software development community.

Chapter 6

Towards a Unified Platform Independent Model for Smart Contracts

6.1 Introduction

Blockchain is an auspicious technology that can support a wide variety of decentralized applications. Smart contracts are the cornerstone beneath a DApp as they allow user to define their own transaction logic, and ensures that the execution of transactions is compliant. Another quintessential function that smart contracts full-fill in DApp development is providing an interface between a blockchains' architecture that processes transaction requests, and the front-end that allows users to interact with the blockchain. Blockchain technology in turn, provides a safeguarded storage for smart contract and a means to transact. Nowadays a plethora of blockchain platforms with an idiosyncratic architecture exist, despite their common characteristic of storing transaction data in blocks. A smart contract and the blockchain architecture on which it is deployed are intertwined, and as a result the implementation of the smart contract concept varies among blockchain platforms [274].

Regardless of the targeted blockchain platform, developing smart contracts has been proven to be arduous, because existing development tools are primitive and there is still limited on-line support available [288]. The difficulty of developing smart contracts for DApps is further magnified by the fact that the current strand of literature on smart contracts predominantly focuses on Ethereum smart contracts [253], which greatly reduces the portability of the tools, methods and approaches presented in these works for other blockchain platforms. However, these problems are not unique to smart contract development.

Research in the field of software engineering has already faced similar problems, and addressed them by suggesting approaches and methods. Model Driven Engineering approach has been touted as a solution for problems like portability, productivity, and interoperability [133]. When developing software by employing an MDA approach, several models are utilized reflecting the viewpoints from which a system can be developed. At the heart of the MDA approach is the use of computation independent, platform independent and platform specific models to capture all relevant information for software design. Mappings between the models enable the transformation from abstract specifications to code. The abstraction of the viewpoints of a software system allows developers to specify novel software in a manner that can be more readily used for multiple platforms, and reuse commonly shared design patterns.

This chapter is based on a publication: A Validated Meta-Model for Blockchain Smart Contracts, submitted to International Journal on Software and Systems Modeling (SoSyM) by B.J. Butijn, D. A. Tamburri & W. J. A. M. van den Heuvel

These attractive prospects have recently sparked researchers and practitioners to leverage MDA for the development of smart contracts. Unfortunately, some prior work [165, 241, 250] presenting an MDA driven blockchain application development method has not made the models underpinning their approach explicit. As a result such methods cannot be replicated or instantiated. Other studies do not take on a platform independent perspective that can be used for multiple blockchains, or contrary, present the approach from an abstract perspective without specifying a platform specific implementation. This study addresses these issues by presenting a platform independent model for the two blockchain platforms that are most often used for smart contract development: Ethereum and Hyperledger Fabric. Besides providing an overview of the architecture of these two platforms, a meta model for smart contracts, and related concepts is presented. Finally, using a bottom-up approach this research aims to further enhance knowledge on blockchains and foster the reusability of code across platforms by presenting a proto-platform independent model for smart contracts.

Hereafter, this chapter reviews all past work that suggests an MDA driven method to develop DApps. Thereafter, in Section 6.3 we elaborate on the research method used to construct and validate the models that are introduced in Section 6.4.1.2 and 6.4.2. The fourth Section presents a PIM for the Ethereum blockchain, and a meta model for Ethereum smart contracts. Another PIM for Hyperledger Fabric networks is presented in Section 6.4.2, along with a meta model for Fabric concepts and smart contract meta model. Section 6.5 provides a PIM for smart contracts and other coherent concepts. The models and findings are thereafter discussed (Section 6.7) and some conclusions are drawn (Section 6.8).

6.2 Related Work: Model Driven Smart Contract Development

Like traditional software development, developing DApps have been shown to consume vast amount of time and resources. Moreover, developing DApps on top of blockchain platforms has proven requires a significant amount of expertise, while little development tools are available [253]. Seeking to leverage the advantages of MDA for business processes, researchers have introduced several MDA based methods to develop DApps. Boubeta-Puig, Rosa-Bilbao & Mendling present CEPchain in their work [35], a method that supports modeling of smart contract for the Ethereum blockchain. Jurgelaitis et al. [122] present an MDA based method to model Solidity based smart contracts. In an other study [122] they also present an MDA driven approach to develop smart contracts in the GO language for Hyperledger Fabric. A commonality among these works is that they present methods that only cater for the development of smart contracts for a specific platform.

Sousa, Burnay, and Snoeck [241] introduce B-MERODE, a model-driven and artifact-centric approach based on business processes to generate blockchain based information systems. Lorikeet [250] is a Model-Driven Engineering tool for Blockchain-Based Business Process Execution and Asset Management. The tool can generate smart contract code from business processes and data registry models based on model transformations. In their work López-Pintado et al. [165] introduce Caterpillar, a module that generates a smart contract based on a BPMN model. The generated smart contract encapsulates the workflow of the BPMN model.

Ladleif and Weske [150] introduce a unified model that encapsulates components required for smart legal contracts. In three subsequent works [146, 147, 151] the authors expand their prior work by introducing modeling support for data sources that reside outside of the blockchain, for cross chain business process choreographies and process execution based on time. However, their model is geared towards smart legal contracts and does not regard

blockchain architecture from a platform specific view point. Neither are any of the abstract concepts presented in the model mapped to a technical realization of a smart contract.

In their work de Kruijf and Weigand employ Enterprise ontology to enhance the understanding of blockchain technology [67] and smart contracts [142]. The model provides invaluable insights into the inner workings of blockchain technology and smart contracts. Yet again, misses a technical specification for the implementation of the concepts presented in their work. While not being a blockchain, Górski and Bednarski [102] present a MDA approach to developing DApps for the Corda distributed ledger.

The studies discussed here provide invaluable insights into how the transaction logic of smart contract could be specified in a domain-specific language. An inherent implication of this approach is that it is not portable to other platforms. Contrary, the MDA approaches discussed here do not present an implementable platform specific viewpoint of smart contracts. Besides these limitations a fundamental model about the infrastructure underpinning smart contracts is missing. However, smart contracts are supported by a blockchain infrastructure and thus cannot be regarded in isolation [94]. Understanding the relation between smart contracts and their environment is therefore crucial for DApp development.

Future research endeavours would greatly benefit from a further specification of the architectures facilitating smart contracts, and a clarification of the relation between smart contracts and the architecture itself. Towards this end our work presents a Platform specific model (PSM) of the architecture for the two largest blockchain platforms: Ethereum and Hyperledger. Besides providing a platform specific viewpoint of the architecture for these two platforms, metamodels for the smart contracts for each platform respectively. To provide researchers and scholars a platform independent viewpoint we provide a unified platform independent model for smart contract development. Contrary to previous research we take on a bottom-up approach in attaining our platform independent model (PIM). That is, we build our unified platform independent model by identifying commonalities between the platforms derived from the platform specific models.

6.3 Research Methodology

This research set out to create three artefacts: (1) a platform specific metamodel for Hyperledger smart contracts, (2) a platform specific metamodel for Ethereum smart contract, both in the context of their blockchain. (3) A metamodel for smart contract development that is platform agnostic. The models are modeled in the UML as it provides a meta-language to support the stipulation of other languages such as the ones we define for each platform. One of the drawbacks of UML is that it offers little support for class related constraints. We remedy this problem by defining class related constraints in the OCL [196]. To develop the platform specific metamodel for Hyperledger and Ethereum we aim to answer the following research questions:

RQ1: What concepts are used to develop Ethereum smart contracts?

RQ2: What concepts are used to develop Hyperledger Fabric smart contracts?

RQ3: How can smart contracts be modelled in a platform agnostic manner?

There is no uniformly accepted method to develop metamodels to our best of knowledge. However, it has been broadly argued that domain ontologies satisfy the criteria for being models, yet are specializations in an object oriented sense. In fact, some authors have even equated domain ontologies to models [111]. Given that ontologies and metamodels are akin, and research in the field of domain ontologies offers several well-established methods to models we adopt such a method. More specifically, in building and evaluating of the models

we employed a "collaborative" method suggested by Holsapple and Joshi [114] where several domain experts collaborate with the designer of the ontology for its development, and a Delphi method [69] is employed evaluation purposes. The method encompasses four phases that are depicted in Figure 6.1.

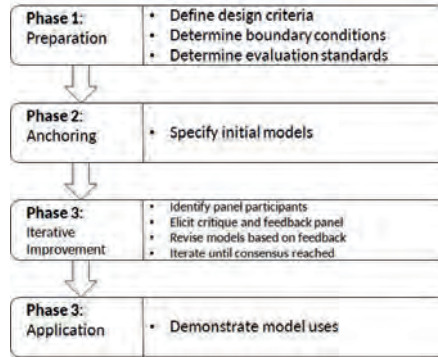


Figure 6.1: Delphi methodology. Adopted from [114]

6.3.1 Phase 1: Preparation

During the first phase of the method we defined the design criteria, boundary conditions and the evaluation standards for the models. In line with Holsapple and Joshi the following design criteria have been selected for the metamodels:

1. Clarity - Whether the concepts are familiar.
2. Comprehensiveness - To measure whether all concepts are covered in the model.
3. Correctness - That the classes, properties and the relation among them are correctly modeled.
4. Consistency - Means that the model does not include any contradictions.
5. Conciseness - As a criterion means that there are no irrelevant concepts included in the model.

Models or ontologies aim to capture the knowledge on a domain or a system within certain boundary conditions to which the design criteria are applied. In this research we defined the domain for each model to smart contracts and required related concepts, rather than blockchain in general. We set another boundary condition by limiting the scope of the metamodels to smart contracts on the Ethereum or Hyperledger platform. A related boundary condition that we set is that the platform independent metamodel should only encompass concepts required to generate a PSM for smart contracts on each of the respective platforms. We proceed from a bottom-up fashion with several layers of depth. The evaluation standards served as a benchmark to compare the outcomes of the development process with. These standards included a set of other related works, and especially the concepts proposed in these models. Standards were further shaped by critiques and feedback from the next phase.

6.3.2 Phase 2: Anchoring

During the Anchoring phase of the study, we specified an initial version of the Hyperledger and Ethereum PSM's. The method proposed by Holsapple and Joshi [114] does not provide

any guidelines as to how to develop an initial version of an ontology. Therefore we followed the steps suggested by Noy and McGuinness [191] to design our PSM models:

1. Determine the domain and scope of the ontology.
2. Consider reusing existing ontologies.
3. Enumerate important terms in the ontology.
4. Define the classes and the class hierarchy following a bottom up or top down approach.
5. Define the properties of classes—slots.
6. Define the facets of the slots (slot value type).
7. Create instances.

The first step suggested by Noy and McGuinness is to determine the domain and scope of the ontology which is akin to setting its boundary conditions. Hence, for this study we skipped this step as they have already been set in the first phase (Preparation). Thereafter as a second step, we considered how concepts of existing ontologies could be reused for a the development of the domain ontology. Drawing from other related works, [7, 274, 285], the documentation of the respective blockchain platforms [61, 119], we identified relevant concepts. This step helped to compare and contrast the ontology against the evaluation standards. Furthermore, the process aided in conducting the third step: the enumeration of important terms. Following a bottom-up approach classes were created and a hierarchy between these classes was established for each of the two platforms. After the creation of the classes for each class its properties were defined. As a fifth step the facets of the slots were defined. Finally, the last step suggested by Noy and McGuinness [191] is the creation of instances of the ontology. This final step serves to verify that the classes and terminology are suitable and sufficient to model the domain.

6.3.3 Phase 3: Iterative Improvement

Two important subsequent steps in Design Science are (1) a rigorous evaluation of the artefact(s) and (2) the refinement of the artefact based on the feedback attained through the evaluation of the artefact. In line with the method proposed by Holsapple and Joshi a Delphi method was employed to gather feedback from domain experts to refine and evaluate the models.

6.3.3.1 Participant selection

When employing a Delphi method to gather data a specific emphasis must be placed on the selection of the panellists as the method strongly relies on the expertise of the participants [69]. The participants selected for our study are 11 senior blockchain developers that are familiar with blockchain architecture and have hands-on experience in developing DApps. In selecting the participants for the panel we used the following selection criteria:

1. The developer has developed at least one Decentralized app for the Ethereum or Hyperledger platform respectively.
2. The developer has deployed at least a smart contract on either the Ethereum blockchain platform or a Hyperledger Fabric network.
3. The developer has attained a certificate, degree or other formal evidence proving that he or she is educated about the architecture of Hyperledger Fabric or the Ethereum architecture.

Participants were approached on the basis of whether they were acquainted with the author of this dissertation. To complement the pool of participants, selected participants were

allowed to recommend potential other participants. After careful consideration and selection based on the selection criteria a total of 3 additional participants were admitted to the panel. In Table 6.1 some descriptive statistics of the participants can be found.

Table 6.1: Descriptive statistics respondents

| | Ethereum | Hyperledger Fabric |
|--------------------------------|----------|--------------------|
| Number of developers | 7 | 4 |
| Avg. years experience | 4 | 3 |
| Avg. Smart contracts developed | 11 | 8 |

6.3.3.2 Data gathering and analysis

When conducting their research Holsapple and Joshi presented their panelists with a full drawing of the ontology they created. Instead of presenting metamodels, for our research panellist were provided a list of concepts related to the blockchain platform they were questioned about.

The Delphi study itself was organized in iterative consecutive rounds. During each round a survey was administered to the panelists to gather feedback on the concepts used for the metamodels. An example of the survey administered for each round can be found in Appendix E. In the first round of the Delphi study a list containing concepts extracted from the extant literature was presented to the respondents. Thereafter a total of 3 rounds continued until no new concepts were identified by the panellists and thus a consensus was reached. An updated list accompanied with a questionnaire was thereafter administered during consecutive rounds.

Several questions were included in the survey to verify whether the concepts were in line with the design criteria. First, to verify whether the concepts in the list satisfied the *clarity* design criterion panellists were asked whether concepts are unclear to them. Another question was included to verify the *comprehensiveness* of the current concept list that asked respondents to indicate whether there were any concepts missing. Respondents that indicated that a concept was missing where asked to provide a description of the missing concept. Suggestions made by respondents to include a concept in the list were compared to suggestions made by other respondents. The suggested concepts where then included in the list of concepts for a consecutive round.

The *correctness* of the list of concepts a question was in the survey allowing the respondents to indicate how sure they are that a specific concepts belongs to the model. This feedback was then processed by further investigating the proposed omission and comparing whether a similar suggestion was made by other respondents. Finally, to address *conciseness* and *consistency* criterion of the list of concepts we asked respondents to point out concepts that were aliases. Identified aliases where omitted from the list when two panellists suggested the same alias.

The concepts identified through the Delphi were used to construct two meta models encompassing all of the concepts relevant to smart contracts on the Hyperledger Fabric or Ethereum platform. These models will be presented hereafter. By synthesizing the commonalities between concepts of both platforms the two models were then compared. Similar concepts were placed in a table until all concepts were checked. Using the concept that both blockchain platforms have in common a platform independent model was constructed that is presented in Section 6.5.

6.4 Platform Specific Perspectives On Smart Contract Platforms

Nowadays several blockchain platforms exist that support smart contracts. Ethereum is the most widely used public network that introduced blockchain based smart contracts is Ethereum [274]. On the other hand, Hyperledger Fabric blockchains are the preferred blockchain in a private setting. For both platforms hereafter an architectural overview is presented as UML class diagrams [197]. Note that in the models several Enum types are used. For brevity's sake we have not included these in the model but included these in the Appendix F.

6.4.1 A Platform Specific Perspective of Ethereum

Ethereum is the most widely used blockchain platform for smart contracts [274]. We will first present an overview of its architecture hereafter. Figure 6.2 depicts Ethereum's blockchain architecture from an overall architectural perspective, and denotes the relations between the elements. In section 6.4.1.2 we further elaborate on the elements contained within an Ethereum smart contract metamodel.

6.4.1.1 Ethereum Blockchain Architecture

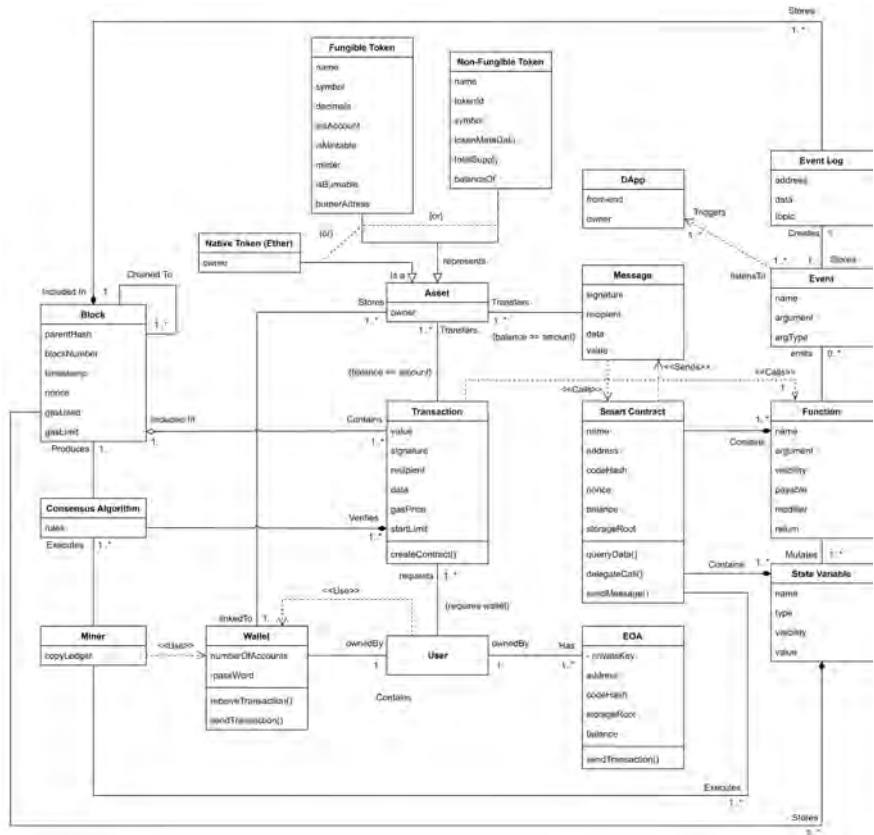


Figure 6.2: A model of Ethereum blockchain architecture

Ethereum offers user accounts, referred to as Externally Owned Accounts (EOA), that are controlled using private keys which have an Ether balance [61]. To send transactions users can make use of dedicated software called a wallet. A wallet is a convenient user friendly interface to manage and store several accounts. One password provides access to each account and therefore also to the balances of each account.

Transactions transfer the ownership of a digital asset from a sender to a recipient. There are three types of digital assets, Ether the native token (crypto currency) of the Ethereum platform, a Fungible token and a Non-Fungible Token (NFT)¹. Cryptocurrency directly represents monetary value. Indeed, there are several exchanges available that allow for the exchange of fiat currency (e.g. Euro or Dollar) and cryptocurrency². For fungible tokens the worth of one token always has the same worth as another token (e.g. one Dollar is equal to another), whereas NFTs represent unique assets such as art that do not have a one-to-one equality. To ensure the quality, security and interoperability of tokens, the Enterprise Ethereum Alliance (related to Ethereum) has suggested standards such as ERC20 and ERC721 for fungible and NFTs [257]. Fungible and Non-Fungible tokens need their own smart contract. Several attributes need to be specified for a token to be compliant with the ERC20 and ERC721 standards: a name and symbol are used to identify the token, where non-fungible tokens additionally need an unique id. As fungible tokens do not represent unique assets it is possible to mint them to increase the supply, or to burn them to achieve the opposite. For non-fungible tokens this is not possible as these tokens represent unique assets. The total supply of each token denotes the maximum of tokens that can be distributed.

A key characteristic of blockchain technology is the use of blocks to store data and safeguard the validity of transactions [285]. Bitcoin and other non-smart contract supporting systems only capture the balance from input accounts to novel-created output accounts in a block. Because smart contracts can invoke multiple transactions Ethereum blocks also store the cumulative effect of transactions captured in other blocks. In other words, Ethereum blocks capture not only transactions performed by the contracts but also the final states that were produced while carrying out these contracts. The smart contracts state variables and associated code are stored in unique contract accounts on the Ethereum blockchain in bytecode format. Contract accounts also have an Ether balance. In essence, therefore the contract account is the smart contract.

The execution of smart contracts can be triggered in two ways: A transaction that has been send by a user from an EOA account, or by messages (calls) that have been send from other contracts. Each transaction includes at least the recipients address, a signature of the sender, the amount to be transferred, and optionally data. In addition, transactions contain a `startLimit` value that express the maximum number of computational steps the transaction is allowed to execute, and the coherent `gasPrice` value that represents the fee a sender is willing to commit in order these steps among things.

Each *gas* unit committed allows for the execution of an atomic instruction that reflects a computational step [61, 163]. Gas was introduced as a transaction fee for the execution of smart contracts to make distributed denial of service attacks expensive. On the Ethereum blockchain Smart contracts can send *messages* to one another. Messages from one contract can trigger a function in another contract. Messages can be perceived as function calls in the form of virtual objects that are serialized and not exist other than on the Ethereum Virtual Machine (EVM)[61].

¹Note that it is also possible to create "normal" tokens that do not adhere to the ERC20 and ERC721 standard. However, the use of such tokens is highly discouraged due to safety concerns.

²See for instance www.binance.org or www.idex.market

Ethereum supports several high-level programming languages that are Turing complete to support users in writing their smart contracts. These high-level language smart contracts are then compiled into bytecode that can be executed in the EVM environment, which can therefore be regarded as the core of the Ethereum platform [61]. Only operations and concepts that are part of one of Ethereum's languages can be performed by a smart contract. However, an important feature of each of these languages is the possibility to define functions that can be triggered by sending transactions to the smart contract. These functions in turn, can mutate the state variables defined in the smart contract code. As a means to notify users that a function has been called a smart contract can emit events and write event logs to the blockchain. Decentralized applications can listen to these events and process the data stored in the event logs, creating an interface between the smart contracts activity and the user.

6.4.1.2 Ethereum Smart Contract Model

Smart contracts are written in code that stipulate the desired transaction logic. Solidity is the most popular language for Ethereum smart contract development. It resembles Javascript and is a statically typed language that can be compiled using the EVM. Smart contracts written in Solidity are akin to a class in an object-oriented language. Figure 6.3 depicts a model in the UML that encompasses the classes and their attributes related to a smart contract, inspired by the Solidity documentation (version 0.7.5) and other generic smart contract concepts. We will now discuss the relationship between these classes and their attributes.

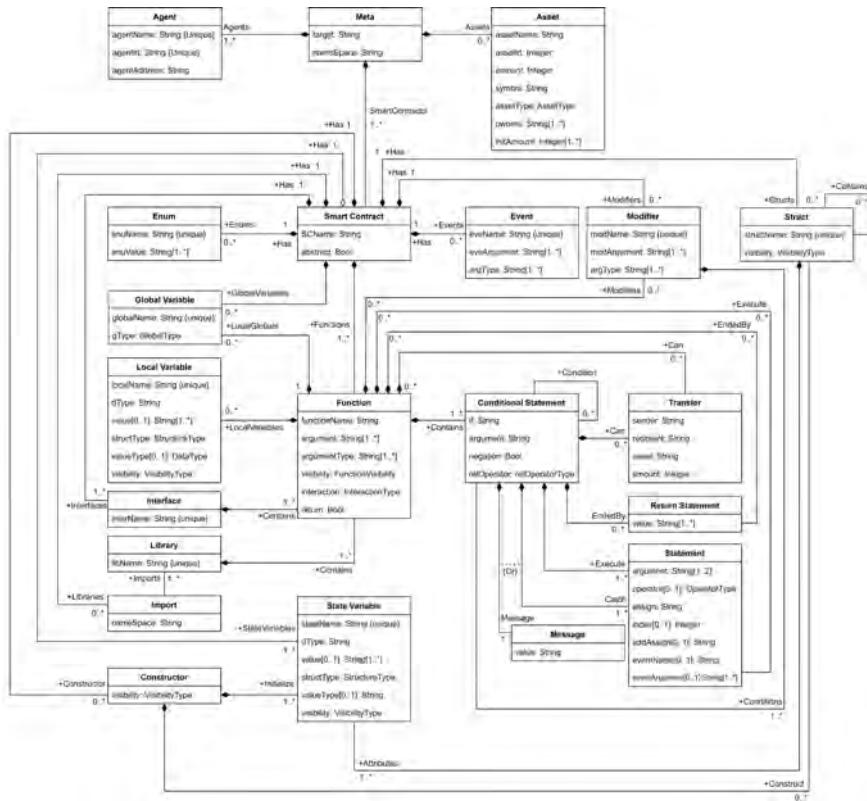


Figure 6.3: A metamodel of Ethereum smart contracts

Meta: Although not native to Ethereum smart contracts a Meta class has been added to the model for Ethereum it has two attributes: `target` and `nameSpace`. The former captures the platform that has been the target of the transformation. This information is important especially for cross-platform transformations. The `nameSpace` here is always public as Ethereum only has a public space.

Agent: Agents are users that own and interact with the smart contract. Within the model agents are included as an element that require an unique `agentName` and `agentId`. The address of the agent is not directly known to the system and needs to be provided by user input.

Asset: From the discussion in the previous section it can be noted that there are two types of standardized tokens to represent assets. We capture this difference using the `assetType` attribute. Other required attributes are `assetId`, `assetName` and a `Symbol` to represent the asset. Some other information might be required such as minters' addresses and so on. Assets are always owned by an agent.

Smart Contract: a smart contract always has a name and holds the several code elements to express transaction logic. These elements are Structs, Enums, Constructor, State Variables, Global Variables, Events, Modifiers and Functions. Other elements are all nested in one of these elements. The exception is however, when the smart contract is abstract in which case the contract can contain only functions, none of which implemented. Other smart contracts can inherit from an abstract smart contract.

Global Variable: are variables used to get information about the blockchain. Examples of global variables are `msg.sender` that returns the sender of a message, and `.now` that returns the timestamp of the current block. The respective data the global variable holds is denoted in the `gType` property. A global variable can be assigned to a variable.

Enum: an Enum is a custom variable type that for which multiple values can be defined. However, it requires at least one value. An unique name is required to declare the Enum variable type.

Struct: akin to an object, in a struct multiple variables are bundled as attributes. A struct needs a name to be instantiated. Besides a name the visibility of the struct must be set in the `visibility` attribute. A struct can have multiple attributes that are declared as variables. Structs can also contain another Struct element.

Constructor: the use of constructor functions is an optional function that initializes the state variables of a smart contract. The function contains several state variables (of any type) that can be set to a particular value. The `visibility` denotes who can access the constructor.

State Variable: A state variable enables the persistent storage of a state on the blockchain. State variables have a `stateName`, `dType`, `Visibility`, `Value` attribute. Each variable must be assigned a name to declare the variable. The `dType` denotes the data type of the variable. Besides standard variable types like integer and string or address, a variable it is also allowed to declare the variable type to be an Enum. State variables declared with a Enum type can only take on the values declared for the Enum type and cannot also be an array. The structure of the variable can be either a standard variable, an array or a Mapping. When mappings are employed the `valueType` attribute also needs to be defined. Visibility specifies who can see the values, the visibility of a state variable is by default public.

Modifier: modifiers guard a function, meaning that a function will not be executed if the conditions in the modifier are not satisfied. A modifier always has a `Name`, `Argument`, and the coherent argument types (`argTypes`). The modifier always contains a Conditional Statement. This condition serves to guard a function and therefore is related to a custom function.

Event: users can be notified using events that again, always have a `eventName`, `argument` and `argType` property. Events are emitted when a function is executed and must be declared within the statements of a function.

Function: functions have a `functionName`, `argument`, `argumentType`, `interaction`, `visibility` and `return` attribute. The name attribute of a function states its name and the argument attribute the arguments required for the function. The `argumentType` attribute stipulates the type of argument. If the function has a modifier, the function will also have a set of Modifier elements contain in Modifiers. The type of interaction that the user can have with the function is captured in the `interaction` attribute with the `InteractionType` enum. Functions are allowed to have three interaction levels: None, Invokable, and Payable. A function is can be ended by a return statement, and execute several statements.

Local Variable: functions can contain several local variables that are used to perform local operations. The values of these variables are not stored in the smart contract. A local variable needs a `localName`, `value`, `visibility`, and `dType`. Similar to a state variable the structure type needs to be known. Operations to local variables are performed using statements within the function.

Conditional Statement: stipulating conditions for a function can be achieved using conditional statements. Solidity supports If statements need to contain at least one variable in the `If` attribute, a relational operator (`RelOperator`) with an argument. A Catch can be defined that contains a statement that is executed when the condition is not satisfied. Alternatively, for when the condition is not satisfied a message can be specified. A conditional statement can contain another conditional statement. To support logical negation the `negation` property can be set to true or false

Message: messages can be stipulated as a fallback mechanism to notify a user calling a function that the operation has failed. The message element is a catch to a conditional statement and only has a string as payload held within the `value` attribute.

Statement: are the lines of code that execute commands. A statement always needs to contain at least an `argument` with a variable name, and a second variable to assign the value of the first variable to. Besides the assignment of the value of a variable, multiple other operation types are possible for Ethereum smart contracts. Whenever an operation is performed, a third variable is required. Standard operations like subtraction, addition, multiplication are supported. These operations are denoted with an `Operator`. For operations to an array the `Index` also needs to be specified. An operations available for mappings that adds a key and value using the optional `addAssign` property. Statements are allowed to contain events, in which case the `eventName` and arguments are required.

Return Statement: a return function is always contained within a function. It returns the value specified for that function which must be either a direct value (e.g. "5") or the name of a variable with an assigned value.

Transfer: transfer is a special function that enables the transfer of an asset between owners. Three fields need to be specified for this function: `Sender`, `Receiver` and the `Amount` to be transferred from what asset.

Library: a library is a data structure that contains only functions. These functions can be used by many contracts. If there are contracts that have common code, then that code can be deployed as library. Each library needs to have a name. Libraries are imported by a smart contract using a statement that imports the library that contains the name.

Interface: Interfaces are expressed using the `interface` keyword and are allowed only to contain non-implemented functions. Each interface needs a name to refer to. Smart contracts may inherit from an interface, interfaces themselves cannot inherit other contracts or interfaces.

6.4.2 A Platform Specific perspective of Hyperledger Fabric blockchains

An important characteristic of public blockchains like Ethereum is that anyone can read and write to the blockchain. However, this characteristic might be less desirable for co-operations that require their transaction information to remain private. To address this need, Hyperledger Fabric was introduced as an alternative to public blockchains for consortia [7]. Figure 6.4 depicts an overview of the architecture of a Hyperledger Fabric blockchain. A more in-depth perspective on Hyperledger smart contracts is discussed in section 6.4.2.2.

6.4.2.1 Hyperledger Fabric Blockchain Architecture

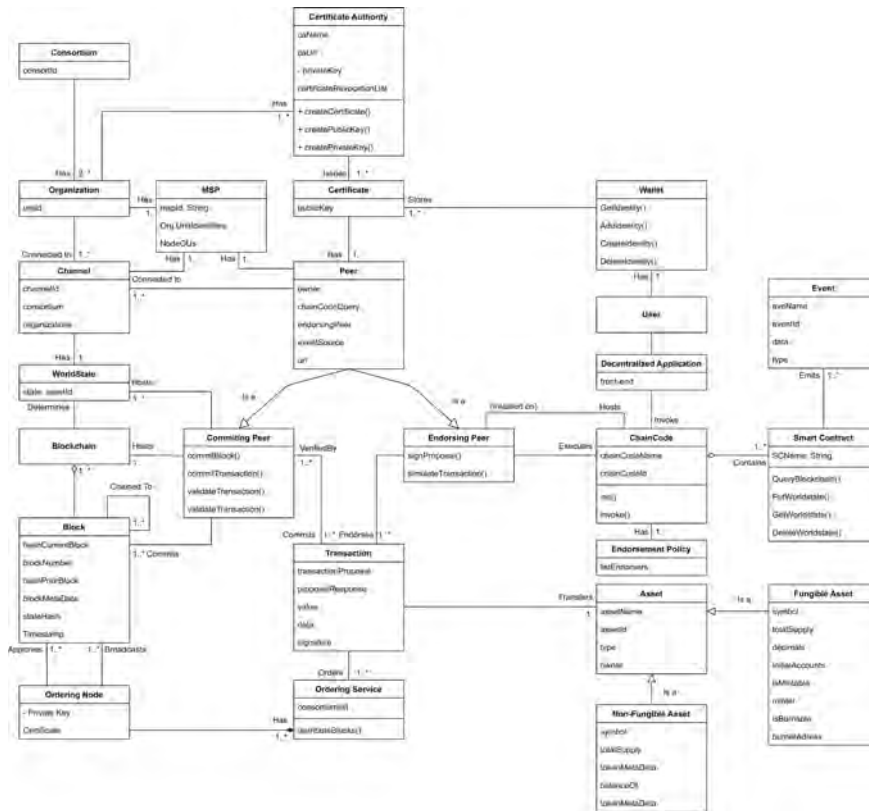


Figure 6.4: A model of the Hyperledger Fabric blockchain architecture

Contrary to the Ethereum platform, Hyperledger Fabric blockchains are characterized as private blockchains that are owned by a cooperation of organizations. Within the network each organization within the network maintains their own Certificate Authority (CA). The CA can be the standard Fabric CA or an already existing CA. By default a CA issues X500 certificates, but other types of certificates are also allowed. Certificates are a paramount feature that sets a Hyperledger Fabric blockchain apart from public blockchains (e.g. Ethereum) as it provides a means to identify users, peers and other nodes in the network. To identify all sub-units of an organization and its members, organizations have a Membership Service Provider.

Two or more organizations can create a consortium and in that capacity, create channels to privately communicate between their organizations. Communication via the channel remains hidden for other organizations within the blockchain network that are not part of the consortium that instantiated the channel. Organizations within the blockchain network deploy peers (nodes) for several tasks. Among these tasks are hosting a copy of the world state, the blockchain, Chain Code or ordering blocks.

Any participant within the network owns assets of which the states are stored in a distributed ledger called the world state. A world state is bound to a channel, and thus only organizations connected to that channel are able to inspect the state of the assets. Because on a

Hyperledger Fabric network the state of each asset is stored as a key value system in the world state, the attributes for each asset can be uniquely defined [7]. Users update the state of their assets by requesting transactions using decentralized applications, also referred to as client applications. By requesting a transaction on a Hyperledger Fabric network Chaincode is invoked. Chaincode is code that can encompass several smart contracts, a implementation that differs from Ethereum smart contracts where the notion of Chain Code does not exist. In a Hyperledger Fabric network the Chaincode encompasses all the concepts required for the execution of the smart contract, whereas in the smart contract the transaction logic is stored.

Upon receipt of a transaction the endorsing peers that host the invoked Chain Code execute it and the smart contract that contains the transaction function. Smart contracts in a Hyperledger Fabric blockchain allow developers to define events that will be emitted once a function is executed. There are several types of events: there are block events that signal the production of a block, transaction events that inform about a transaction request, and events related to smart contracts. The endorsing peer then verifies the outcomes of the execution of the Chaincode and whether the proposed changes to the world state are correct. After signing the transaction the endorsing peer sends back the response to the transaction request to the decentralized application.

The DApp then forwards the signed to an ordering service which packs several transactions into a block. Peers ordering the transactions and broadcasting the blocks are called ordering nodes. Another type of peer, committing peers host the blockchain and commit novel blocks to their version of the blockchain. Besides hosting the blockchain committing peers host a copy of the world state of a channel.

6.4.2.2 Hyperledger Smart Contract Model

Writing Hyperledger Fabric chaincode and smart contracts is supported in Java, JavaScript and Go. To ease the development of decentralized applications built on top of a Hyperledger Fabric network, Hyperledger Composer³ was introduced. Unfortunately now deprecated, Hyperledger Composer was a tool that allows developers to define the resources required for their DApp, and coherent Smart Contracts. Still the concepts proposed for the Hyperledger Composer tool sheds a light on the concepts used when developing smart contracts. After the depreciation of Hyperledger Composer, the Hyperledger Fabric SDK (Software Development Kit) was released with the same purpose. The chaincode generated by the SDK stores a JSON schema for each asset that needs to be managed by the network.

³<https://hyperledger.github.io/composer/latest/index.html>

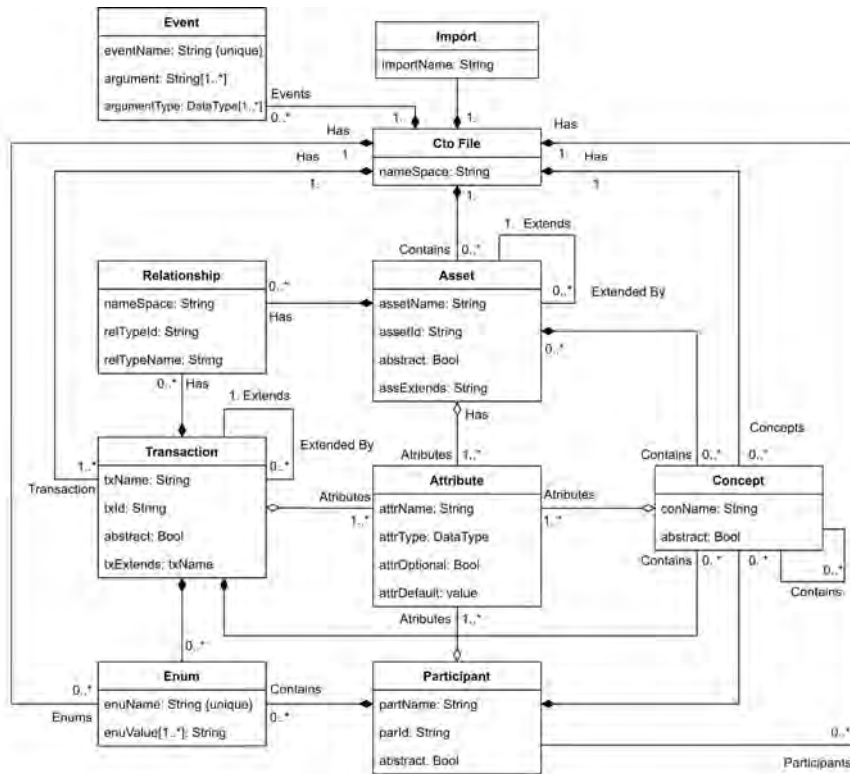


Figure 6.5: An overview of resources for Hyperledger Fabric

Participant: the participant class is used to specify the participants organizations or persons that will use the DApp to govern their transactions. Each participant element needs at least a name (`partName`) and Id to be created. The `partId` attribute is used as a reference in other elements and to identify the participant on the Hyperledger Fabric network. A participant element can be defined as being abstract.

Asset: again, like an Ethereum asset an asset for Hyperledger Fabric needs a name and Id so it can be recognized by the network. An asset can extend other assets to take all attributes and fields, or be abstract itself.

Transaction: within any blockchain network enabling transactions is the main service provided. Similar to assets and participants, transactions require a name and Id, can be abstract or extended by other transactions.

Attribute: albeit that there are pre-defined base classes for Hyperledger Composer, each Transaction, Asset and Participant element are allowed to have user defined attributes. Each attribute requires a name and the data type needs to be specified. If the attribute contains an array this also needs to be specified.

Concept: to specify classes other than participants, assets or transactions concepts are used. Concepts have a name for referencing in other concepts or participants, assets and transactions. An example of a concepts is an address with the attribute street, city etc. Any concept is allowed to be abstract.

Enum: Enums are user defined variable types that have a name and one or more values. An Enum that has been declared is allowed to be used to declare as the type of an attribute.

Event: users can be notified using events that again, always have a `eventName`, `argument` and `argumentType` property. Events are emitted when a function is executed and must be declared within the statements of a function.

Relationships: between all of the aforementioned concepts relations can be defined. The relationships are mappings between the concepts. For all relationships the namespace of the file where the element is defined needs to be known, along with its type. Another important specification is the direct Id of the concept.

Import: an import statement is used to obtain information or function from other files. To specify from which file this information needs to be imported, an `importName` must be given that refers to another file.

In the smart contract, the transaction logic is defined using callable functions. One of the features that sets a Hyperledger Fabric network apart from other platforms is the possibility to define policies who can perform what operations on the functions in the smart contract. Each rule is stored in an Access Control File (ACL) that is managed by the Chaincode. Queries are stored in a separate file.

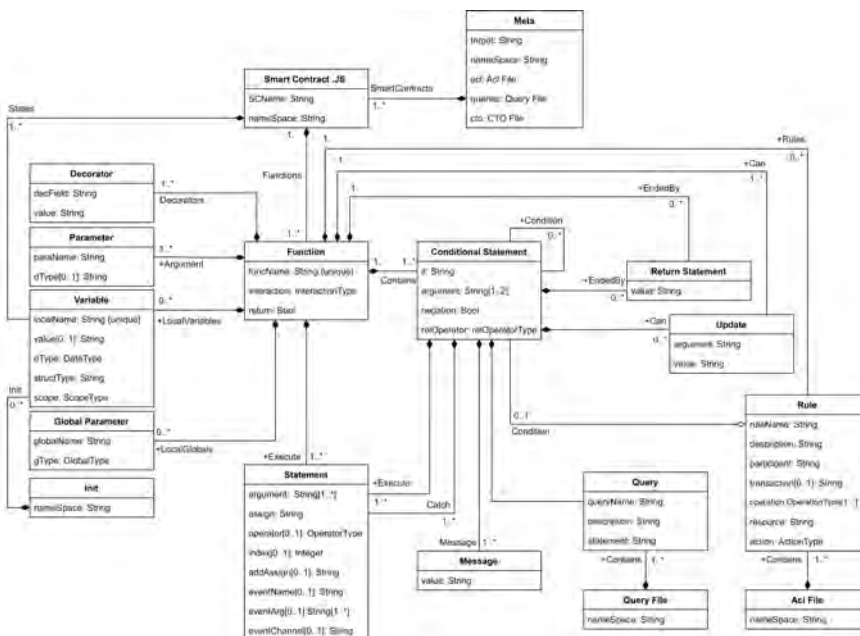


Figure 6.6: A Metamodel of Hyperledger Fabric smart contracts

Meta: For administrative purposes and to bind the CTO, ACL and Query file together a meta element is added that contains all meta information needed for the creation of the models. The `nameSpace` attribute is the most important information that the class provides. This attribute denotes on which of the many chains a consortium network can encompass the smart contracts should be deployed using the model.

Function: within a hyperledger fabric smart contract functions update the states of assets. Each function requires a unique name to prevent function overloading, and an argument that is always equal to the name of the parameter provided by their decorator. Whenever the function has contains a return statement that should be denoted in addition. There are two categories of functions: the ones that are meant to be interacted with by users and

those that are not. To cater for this difference the attribute `interaction` denotes the type interaction possible with the function.

Decorator: functions declared within the smart contract need to have a decorator that decorates the function with the required information and enforces the parameter or return type. Any required information to decorate the function is contained in the decorator field (`decField`). The first information that the decorator field requires is whether the decorator is linked to the parameters or return value. Second, the `value` field needs to contain the name of a parameter or return value that is to be enforced. Finally, a parameter or return value needs to be created with a name to identify it that is linked to a function

Parameter: the parameter generated by the decorator is only used for the function linked to the decorator that created it. The parameter is then connected to the function.

Variable: Unlike the Ethereum blockchain Hyperledger Fabric smart contracts do not have state variables that are directly linked to the contracts. Instead, states are stored separate in the World State. The concept of state variables can be achieved by querying the value of a certain asset or participant and assigning it to a variable. This would require the creation of a query for each state however. Within a function, local variables are allowed to be declared for briefly memorizing values. Local variables need a name and a value to be declared. Any value contained in the local variable is not persevered as data storage. To discern between "states" and local variables the `Scope` attribute is used.

Conditional statement: conditions for the enactment of a transaction are formulated using conditional statements. The `if` within a conditional statement needs to contain a parameter name (`paraName`), additionally an relational operator can be added in which case an argument is required argument. A `catch` is allowed as a fallback when the condition is not satisfied and contain a message.

Query: a query will read the current state of the world state and return it back to the user. Queries are predominantly used to retrieve information about states and as such linked to variables containing this information. To create a query a name for the query needs to be declared, optionally a description of what information the query returns can be included. The statement attribute of the query needs to contain the name of the channel the asset is stored (`netName`) and the name of the name of the asset that the query is performed on (`assetName`).

Global Parameter: a special type of variable is allowed to be declared that bears its own name and is akin to a local variable. Like a local variable a global variable has a value, this value is the result of a query however and therefore includes a `queryName`. Since these local variables only preserve global information about the ledger state a differentiating between a local and global variable is warranted.

Statement: functions and conditional statements may include a statement. A statement needs to have at least an argument, and optionally an `assign` field that contains the name of a local variable or parameter. Hyperledger Fabric smart contracts support various operations to be performed on variables. Whenever these operations are performed a third argument is needed. Some variables are arrays and have multiple values. The `index` in a statement denotes what value or variable name should be inserted at what index, the `addAssign` operation allows to add a new value to the array. Events are emitted using the `emit` keyword, to perform this operation three fields are needed: first the name of the event (`eveName`), the coherent arguments (`eveArgument`) and the name of the channel on which the event is conveyed (`eventChannel`).

Return: conditional statement or functions are allowed to contain a return statement, that can either return a value or when they stipulate the name of a variable the value of this variable.

Update: instead of making a transaction and updating the balance of one or more of the participants, on a Hyperledger Fabric network a transaction updates the states of an asset.

Rule: a rule within the network defines which participants are allowed to call the function and what operations they command when calling the function. A rule has a name as an identifier, a description is optionally added to provide a user friendly description of what the rule does. Participants denoted for a rule can be set to a specific participant by proving the name of that participant (`partName`) or stipulating that the rule encompasses all participants. By providing the name of the transaction (`txnName`) a link can be established between the rule and a function. An operation field defines what operations are allowed to be performed. The set of operations that can be defined are: *Create*, *Read*, *Update*, *Delete* and *All*. Within a rule the resource that it concerns must also be given: for each resource the *nameSpace* must be known and its name. Further specifying the actions allowed for each participant is possible by stating the action that is limited to allow and deny. Rules themselves can include a conditional statement that verifies for instance whether the requester of a transaction is the owner of the asset.

6.5 Towards a Platform Independent Metamodel for Smart Contracts

The results of the Delphi study has been used to construct a platform independent meta model for smart contracts. A PIM encapsulates concepts that are platform-dependent without confining the model to specific technology platform, in this the case blockchain platform. We identified several commonalities and differences between the two platforms that will now first be discussed. Thereafter we present a PIM model that caters for the modeling of platform agnostic smart contracts.

6.5.1 Finding Common Ground

On a Hyperledger Fabric blockchain each agent is part of an organization that manages the identities of agents, this concept is not present on the Ethereum blockchain as it is public. A commonality is that both platforms cater for the registration of agents by providing user accounts that have a public key that acts as an address. Transactions can be requested to transfer assets using a wallet as it stores both an owners private and public keys.

The Ethereum platform has its own native token (Ether) better known as cryptocurrency, and caters for the creation of custom tokens that represent specific assets. The Hyperledger platform does not have any native token (cryptocurrency), but allows its users to create tailor-made assets to represent monetary value. To promote the portability and interoperability of tokens between Hyperledger Fabric blockchains and the Ethereum blockchain the Hyperledger platform has adopted the ERC20 and ERC721 standards. Although users are allowed to deviate from these standards it provides a common ground for the standardization for the representation of assets on both platforms. Regardless of the underlying platform both types of tokens are created by deploying a smart contract.

A noticeable difference between the platforms is that the approaches to store data and create consensus on the validity of transactions varies among the two platforms [274]. Ethereum uses blocks to store the states of smart contracts and other required information, while the states of assets on the Hyperledger Fabric platform are stored on both a blockchain and the world state of a channel. Another important difference is where and how smart contracts are deployed. Ethereum smart contracts are deployed on the respective blockchain by sending its contents with a transaction. The initial agent that deployed the smart contract is its owner. Hyperledger Fabric smart contracts are deployed by installation on a number of nodes and the owner can be anyone. Another distinction between the platforms is that to reduce the

size of the smart contract, Ethereum allows that functions are imported from libraries. Hyperledger does not allow for such imports, in fact Hyperledger Fabric smart contracts are deployed in Docker containers [274] that are isolated.

The manner of interaction between the smart contract and a user is similar for the platforms. Throughout the life-cycle of a smart contract, agents interact with the smart contract to change its state by sending it transactions, that in turn call functions. On the Ethereum platform smart contracts have their own balance and state. Functions can update the balance of a smart contract or change a state, however these are considered two different operations. When calling Hyperledger smart contract functions no distinction is made as in both cases the state is updated through the World State. An interesting observation is that while functions are meant as means for users to interact with a smart contract there are also a large host of functions that are not meant to be interacted with.

Hyperledger and Ethereum smart contracts both include the notion of conditional statements and statements. Whereas most conditions in the smart contract need to be satisfied by agents performing (trans) actions, the statements inside the function that is called by the transaction are executed by the smart contract. Smart contracts can automatically send messages (Ethereum) or transaction proposals (Hyperledger) to transfer ownership of assets. Events are used on both platforms to signal that user action with a smart contract has taken place. Dapps can listen to events and accordingly, display any relevant information when required [45]. Other information about the state of the blockchain is retrieved using queries. Ethereum's Solidity has built-in queries to get global variables like the current time, block size and sender of a message. Within the Hyperledger Fabric framework there are built-in queries to get global variables yet these need to be assigned to a dedicated variable.

The execution of a function restricted using a concept that both platforms have in common called pre-conditions. Preconditions are conditions that must be true before the execution of before the execution of a section of code or a function. When writing Ethereum smart contracts in Solidity these pre-conditions are stipulated using modifiers. Vyper [248] another programming language for Ethereum, also allows writers of smart contracts to state pre-conditions with corresponding post-conditions. In a Hyperledger Fabric smart contract the concept of a precondition is implemented as a rule. However, rules for Hyperledger Fabric smart contracts only govern who can access and perform certain operations (e.g. read and write transaction information). Such patterns are known as access control rules [222]. The types of precondition for Solidity and Vyper is not restricted to access control rules, but can be freely formulated.

However, some patterns for smart contract conditions can be identified. In their work Ladleif and Weske [150] identify three other types of conditions: evaluative conditions, causal conditions and temporal conditions. Evaluative condition compare two variables that might stem from different data sources, and can be used for non-specific patterns. The concept of temporal conditions is that other conditions are bound to a deadline in combination with this type of conditions. Causal conditions denote the casual relations between a condition. However, they fail to include access control related patterns that are also used for Ethereum smart contracts⁴. Thus, we identify access control patterns as a fourth type of precondition.

This raises a problem however, as rules on a Hyperledger blockchain not only govern who can perform an action, but in addition what operation (read, write, delete, create). The principle

⁴For an example of such patterns see Open Zeppelin <https://docs.openzeppelin.com/contracts/2.x/access-control>

of the Ethereum blockchain is that once created assets or transactions cannot be deleted⁵, nor can they be just created. This is at odds with the aforementioned delete and create operations that a Hyperledger blockchain allows for.

6.5.2 A Platform Independent Model for Smart Contracts

Based on the commonalities and differences discussed in the previous paragraph we constructed a PIM model that allows for the development of smart contracts in a platform specific context. The PIM model requires some specification of meta-data like the targeted platform. On both platforms agents have an address, yet the implementation of this concept differs and depends on the platform. To cater for this difference, users need to define the address of the agent on the Ethereum blockchain or the certificate and nameSpace on the Hyperledger private blockchain. These will then be incorporated into the model as meta-data. Fig. 6.7 portrays the elements that the model encompasses that will now be discussed. Enums depicted in the model can be found in Appendix F.3.

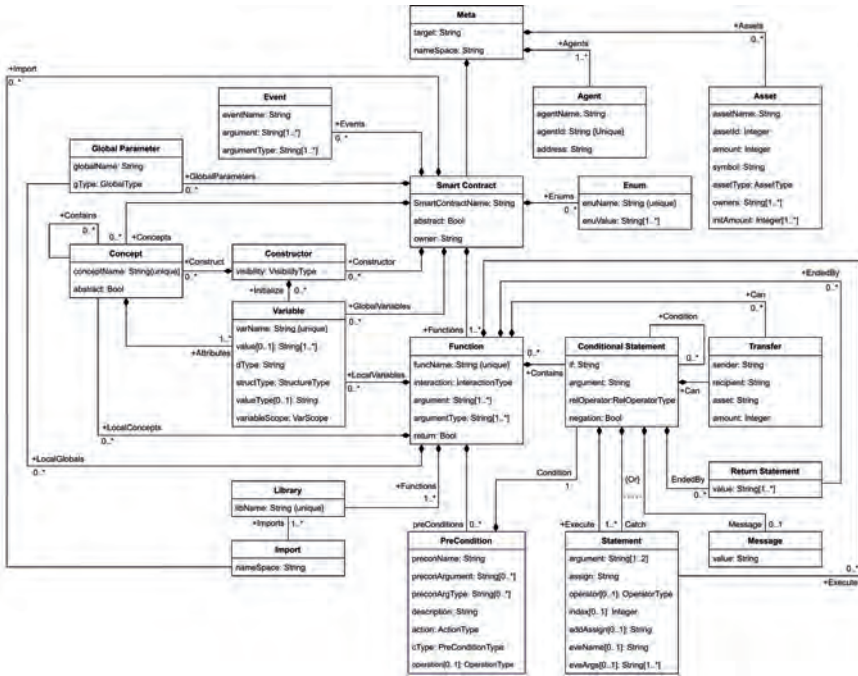


Figure 6.7: A platform independent model for smart contracts

Meta: Again, the first element included in the model is Meta, that has two attributes: the attribute `target` contains the platform that is targetted for the transformation. The `nameSpace` property captures where the smart contracts are deployed. When the target of the transformation is the Ethereum platform this could denote any string, whereas if the Hyperledger Fabric platform is targetted the `nameSpace` contains the blockchain within the consortium network where it should be deployed. The meta class also binds the agents and assets with the smart contracts into one model.

⁵This statement needs qualification as smart contract on the Ethereum blockchain actually can selfdestruct. However, any transaction and the smart contract itself remain present on the blockchain.

Asset: The safe and disintermediated exchange of assets the primary *raison d'être* of blockchains. Standards like the ERC20 and ERC721 provide a common manner to represent assets in a platform agnostic manner. The prerequisite knowledge is that it is known whether the asset is unique or not, which is denoted using the `AssetType` attribute. In line with these standards the `assetName`, and initial amount of the asset must be known. Agents can own portions of fungible tokens and the `owners` field is used to denote the list of initial owners. The `initAmounts` attribute must contain an equal number of amounts that correspond to these owners. For traceability purposes the asset will have a unique id `assetId`. Further information that is required to adhere to the standards needs to be provided by the user.

Agent: Another common concept is that of agents. Agents are the owners of assets and the entities that interact with the smart contract. All agents have an `agentName` attribute, and unique `agentId`.

Smart Contract: A first element that obviously needs to be included is a smart contract. For the PIM the smart contract needs the attribute `SmartContractName` as string. To avoid collision with other smart contracts the name needs to be unique. The attribute `abstract` denotes whether the smart contract is abstract or not.

Global Parameter: Global parameters are used in the PIM as means to retrieve information about the platform. The name attribute is used to declare the variable name. Users can stipulate what type of information they want the global variable to contain using the `GlobalType` attribute. The type of information is limited to `currentTime`, `valueTransaction`, `senderTransaction`. On the Ethereum blockchain all global information is stored in on one chain. When using Hyperledger multiple private chains can exist and to get any global parameter the `Namespace` needs to be known during translation.

Event: As both platforms include the notion of events this element is included into the meta-model. Based on the similarities between the platforms it has the properties of `eventName`, one or more `argument` with a coherent equal amount of arguments in the `argumentType` attribute.

Enum: Enums are another common concept that allow for a restricted set of values. An Enum has an `enumName` attribute that denotes the name of the enum. Like on Hyperledger blockchains and Ethereum the `enumValue` property contains the set of values the enum is allowed to have.

Variable: Variables have a unique name in string format. A variable is optionally declared with a value. When drafting Ethereum smart contracts, a distinction must be made between state variables and local variables. Hyperledger does not make this distinction but states are captured in the World State that can be queried. To cater for this difference the general concept of variable is included that has a specific scope (`variableScope`); Local or Global. Local variables must be declared within a function while global variables can be declared anywhere. The type of variable denotes whether it is a primitive type or an enum, in which case the name of the Enum must be declared and must correspond with a `structType` of Enum. The `valueType` attribute is optional and only used when the `structType` attribute is equal to `KeyValue`.

Concept: Concepts represent "things" within a smart contract and are used to define the concepts of Structs (Ethereum) or Concepts on a Hyperledger Fabric blockchain. Platform agnostic concepts are named using the `conceptName` attribute. Each concept in turn, includes several variables that constitute to the attributes of the concept. Concepts can be nested and include another concept.

Constructor: The constructor is an integral part of the smart contract that sets the initial values (if any) for variables, and concepts. It only contains variables and concepts in the form of (`varName`) and coherent values, and Concepts.

PreCondition: The discussion in the previous section highlights that both platforms have a concept that strongly resembles preconditions, yet the types of conditions differ. To cater for

this difference, the type of precondition is restricted to the Evaluative, Temporal, Causal and Access Control specified in the `cType` attribute. Each `PreCondition` contains one conditional statement that follows the patterns for the respective type of `PreCondition`. Depending on the precondition type different variables are used for the condition. Whenever the type of the `PreCondition` is Access Control and the targetted platform is Ethereum, only read and update restrictions may be denoted in the `operation` attribute.

Function: Functions are arguably one of the most important concepts as they allow for interaction of a user with the smart contract. From the comparison between platforms it can be observed that there are different kinds of interaction possible with functions ranging from none, invocable to allowing for the exchange of assets. The property `interaction` again captures these options. Besides the type of interaction, the function name must be known for which the `funcName` property caters. The arguments of the function and coherent argument types can be captured in the `argument` and `argumentType` attributes respectively. Functions can be associated with none or several preconditions.

Condition: Conditions are shared concept, that always include a statement. The conditional statement always needs an `if` attribute requires a value or the name of a variable. The relational operator (`relOperator`) stipulates the comparison for the condition with the argument. A catch can include either a statement or a message.

Message: A message has one attribute called `value` that holds a string with a message for when the execution of a conditional statement has failed.

Statement: Statement are also included as a concept as this is the main manner to let the smart contract execute commands other than transfers. From the comparison of the two blockchains it follows that a statement is used to emit events and perform actions. Thus, to represent this a statement must contain at least the name of an event to emit or an argument.

Transfer: The transfers of assets is commanded by the transfer statement. Hyperledger smart contracts do not transfer an asset but rather update the ownership of an asset. We adopt the transfer statement as means to signal what is transferred to whom. For the execution of the transfer we need information on sender, recipient, amount, and which asset is being transferred.

Return: In line with the models for Ethereum and Hyperledger a return statement concludes a function. It will only have the `value` property that contain either a value or the name of a variable.

Import: The Import class has as a sole function to import a library the `nameSpace` property functions as an information holder where the file with the library is stored.

Library: We cater for the possibility to import libraries by including the Library class in the model. It has a dedicated name stored in the `libName` attribute and has several functions attached to it.

Hereunder some constraints specified in the OCL for the Smart Contract, Global Parameter and Conditional Statement classes of the platform independent model. Each constraint is related to a class (`context`), has a name (e.g. `SmartContract`), with the invariant specifying the exact constraint (`inv`) that must hold. For brevity's sake However, all constraints related to the models can be found in Appendix G.

```
context SmartContract
  inv: scPIM!SmartContract.allInstancesFrom('scPIM')->isUnique(SCName)
      and SCName->size()>0
  inv: Functions->notEmpty()
  inv: owner->size()>0
  inv: Functions->notEmpty()
```

In line with the explanation in section 6.5.2, elements representing a platform independent smart contract require a unique name that has a length of at least one. The first invariant ensures this property by checking the name of the smart contract against other known smart contracts. All smart contracts also need at least one function which the second invariant ensures by stating that the Functions collection cannot be empty. Finally, each smart contract must have an owner (third constraint).

```
context GlobalParameter
  inv: scPIM!GlobalParameter.allInstancesFrom('scPIM')->
    isUnique(globalName) and globalName->size()>0
  inv: Set{'CurrentTime', 'SenderTransaction',
    'ValueTransaction'}->includes(gType)
```

Global parameter elements again need a unique name with a length of at least 1. The attribute global type (gType) can only be a specific set of values, in this case "CurrentTime", "SenderTransaction" and "ValueTransaction".

```
context Conditional Statement
  inv: scPIM!Variable.allInstancesFrom('scPIM')->exists(v | v.varName = if)
    xor scPIM!GlobalParameter.allInstancesFrom('scPIM')->exists(gp |
      gp.globalName = if)
    xor scPIM!Agent.allInstancesFrom('scPIM')->exists(ag | ag.agentId =
      if)
  inv: scPIM!Variable.allInstancesFrom('scPIM')->exists(v | v.varName =
    argument)
    xor scPIM!GlobalParameter.allInstancesFrom('scPIM')->exists(gp |
      gp.globalName = argument)
    xor scPIM!Agent.allInstancesFrom('scPIM')->exists(ag | ag.agentId =
      argument)
  inv: Set{'greaterThan', 'equalTo', 'smallerThan',
    'notEqualTo'}->includes(relOperator)
  inv: Set{'true', 'false'}->includes(negation)
  inv: Catch->notEmpty() implies Message->isEmpty()
  inv: Message->notEmpty() implies Catch->isEmpty()
```

Constraints for conditional statements elements are more complex. The if attribute can only contain an existing variable name, global parameter name, or agent id. An argument captured in the argument attribute must adhere to the same constraint. The purpose of these first two constraints is to ensure that conditional statements are not declared with non-existing variables. Whenever a conditional statement is false either some code is executed or a message is send. The last two constraints ensure that if a message is specified no statement is executed and vice versa.

6.5.3 Transformation Rules

The transformations between the PIM and PSM's are enabled using transformation rules (a.k.a. mappings). Transformation rules consist of an input pattern introduced by the keyword *From* and an output pattern with the keyword *To*. A source pattern specifies which part of the PIM maps to what target pattern indicated with the word *out*. The implementation of the targeted pattern shows the specifics of the transformation. This specification

may contain one or several pattern elements. A target pattern consists of a variable declaration and a set of bindings (assignments) shown with the arrows. Listing 6.1 encompasses the transformation rules for the transformation of the PIM smart contract to a Ethereum smart contract. A complete overview of all transformation rules to transform a PIM to a Ethereum PSM can be found in Appendix G.

```

1 lazy rule SC2SC{
2   from sc : scPIM!SmartContract2SmartContract
3   to
4     ssc : solPSM!SmartContract(
5       SCName <-sc.SmartContractName,
6       abstract <-sc.abstract,
7       owner <-sc.owner,
8       Interfaces <-sc.
9       Imports <-sc.
10      Enums <-sc.Enums->collect(el | thisModule.createEnum(el)),
11      StateVariables <-sc.GlobalVariables->collect(el | thisModule.
12        createStateVariable(el)),
13      GlobalVariables <-sc.GlobalParameters->collect(el | thisModule.
14        createGlobalVariable(el)),
15      Structs <-sc.Concepts->collect(el | thisModule.createStruct(el)),
16      Events <-sc.Events->collect(el | thisModule.createEvent(el)),
17      Modifiers <-sc.getModifiers(),
18      Functions <-sc.Functions->collect(el | thisModule.createFunction(el))
19      Constructor<-thisModule.setConstructor(sc.Constructor)
20    )
21 }

```

Listing 6.1: Transformation rules to transform platform independent smart contract to an Ethereum smart contract

The transformation rule in Listing 6.2 delineates how functions are in a user specified PIM are transformed to a function for a Hyperledger smart contract. Appendix G presents all other transformation rules for a transformation from the PIM for smart contracts to a Hyperledger smart contract model.

```

1 lazy rule createFunction{
2   from
3     fu : scPIM!Function
4   using
5     otherPrecon: Sequence(scPIM!PreCondition) =
6     select(pre | pre.cType = 'Temporal' or pre.cType = 'Evaluative' or pre.
7       cType = 'Causal')
8   to
9     hfu : hyperPSM!solFunction(
10      funcName<-fu.funcName,
11      return<-fu.return,
12      interaction<-fu.interaction->setInterActionType(fu.interaction),
13      Argument<-createParameters(fu.argument, fu.argumentType),
14      EndedBy<-fu.EndedBy->collect(el | thisModule.createReturnStatement(el))
15      Decorators<-createDecorators(),
16      Rules<-fu.preConditions->select(pre | pre.cType = 'AccessControl')->
17        collect(el | thisModule.createRule(el)),
18      LocalVariables<-fu.LocalVariables->collect(el | thisModule.
19        createStateVariable(el)),
20      LocalGlobals<-fu.LocalGlobals->collect(el | thisModule.
21        createGlobalParameter(el)),
22      Contains<-thisModule.createConditions(fu.Contains, otherPrecon),
23      Execute<-fu.Execute->collect(el | thisModule.createStatement(el)),
24      Can<-fu.Can->collect(el | thisModule.createUpdate(el))
25    )
26 }

```

Listing 6.2: Transformation rules to transform platform independent function element to an Hyperledger Fabric smart contract function

Only two examples transformation rules have been presented here. While these rules partially reveal how a transformation from a PIM to a PSM takes place, some more clarification

is needed to fully understand the interplay between the transformation rules. In the section hereafter we will further expound on how the actual transformation takes place based on these rules.

6.6 Smart Contract Generation

So far the artefacts required to model smart contracts have been discussed. However, the transformation from a PIM to a PSM, and ultimately to the platform specific code, requires a tool. The tool is written in TypeScript [32] a superset of JavaScript the standard programming language for the world wide web⁶. TypeScript allows for static typing during compilation to ensure that only the appropriate data type is assigned to a variable, and allows users to define the shape of objects. The initial input for the tool, and the intermediate results of the model transformations are stored in JSON format, to ensure that they are transferable between the generated models. The activity diagram in Figure 6.8 portrays the workflow of the smart contract generator.

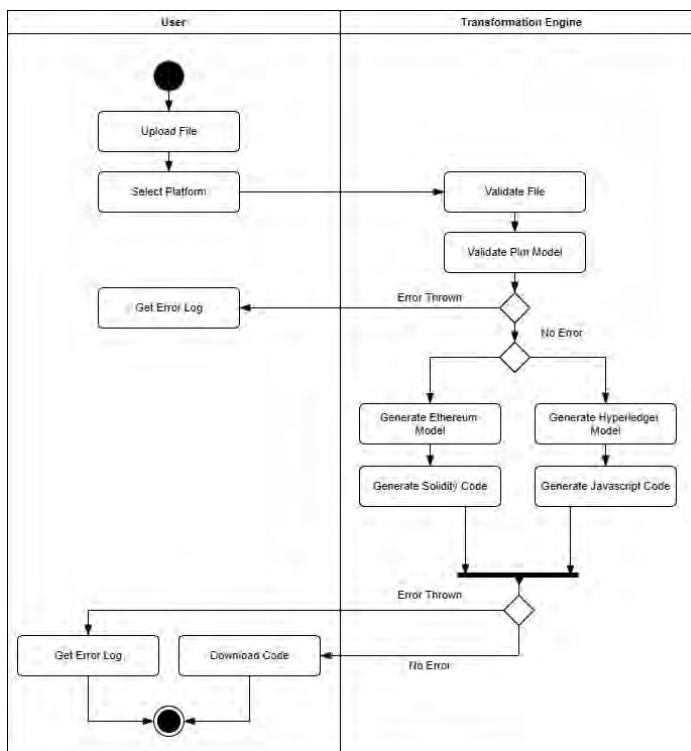


Figure 6.8: Algorithmic steps for Smart Contract generation

6.6.1 Validation Platform Independent Model

The input for the tool is a JSON formatted file that pertains all information required to create the smart contract for a platform. Thus, when the smart contract generation the tool is used

⁶For the sake of replicability all code related to the tool is available as online supplement <https://github.com/BJBut/TransformSC>

it expects a JSON file as input. To guard against errors created by the wrong file type it will validate whether an input has the correct file extension (.JSON). Thereafter it will analyze the correctness of the input file in terms of syntax.

TypeScript ensures run time verification of types, however it cannot do so from external input. Therefore *ex ante* validation of the JSON file containing data on the PIM is required before commencing the transformation from PIM to PSM. The JSON schema language [210] is employed for JSON instance validation in combination with the AJV library⁷. A JSON schema is written in the JSON syntax and is programming language agnostic. Its vocabulary caters for the definition of objects, their attributes, coherent data types and constraints. Combined, this info results in a schema that contains info on the shape and conditions of objects, arrays and standard variables. In turn, schema's enforce type checking of variables, but also for more complex checks such as whether an object has all required properties. Listing 6.3 depicts an example of a schema for a Variable, that enforces and objects' property types, and related other constraints.

```

1  VariableDef = {
2    type: "object",
3    required: ["Variable"],
4    properties: {
5      Variable: {
6        type: "object",
7        required: ["varName", "dType", "structType", "variableScope"],
8        properties: {
9          //@dev: the isUnique attribute in the schema ensures that
          //       the variable name is unique.
10         "varName": { type: "string", minLength: 1,
11                     "isUnique": {"$data": "1/PimRegister/varNames"} },
12         "value": { type: "array", minItems: 1},
13         "structType": { enum: ["Enum", "Variable", "KeyValue", "Array"] },
14         "variableScope": { enum: ["Global", "Local"] }
15       },
16       //@dev: these are the constraints related to the Variable
17       //       object.
18       dependencies: {
19         "structType": {
20           oneOf: [
21             {
22               "properties": {
23                 "structType": { "const": "KeyValue" },
24                 "dType": { enum: ["String", "Byte", "Integer", "Address"] }
25               },
26               required: ["valueType"]
27             },
28             {
29               "properties": {
30                 "structType": { "const": "Array" },
31                 "dType": { enum: ["String", "Byte", "Integer", "Address"] }
32               },
33             },
34             {
35               "properties": {
36                 "structType": { "const": "Enum" },
37                 "dType": { type: "string" }
38               },
39             },
40           ],
41         },
42       }
43     }
44   }

```

⁷<https://ajv.js.org>

```

43
44
45         "properties": {
46             "structType": { "const": "Variable" },
47             "dType": { enum: ["String", "Byte", "
48                 Integer", "Address"] }
49         },
50     ],
51 }
52 }
53 }
54 }
55 }

```

Listing 6.3: Example schema variables

Besides testing for supported classes and their attributes, the validator checks data type discrepancies such as when the datatype of a variable is declared to be a "string" while its value is an integer. When a discrepancy is detected the validator will log an error and halt the generation process. Using JSON schema it is also checked whether for instance a smart contract contains at least one supporting element. Duplicate function, smart contract, precondition and variables with a global scope names are validated with the *isUnique* keyword to ensure their uniqueness. Variables with a local scope that stem from different functions are exempted and will not throw an error. After passing the validations, the creation process for a PSM that the user chose commences. The choice for a platform is expected upon the validation of the JSON file, along with any information related to agent addresses.

6.6.2 Ethereum PSM Creation and Code Generation

The tool provides support for smart contract code generation in Solidity based on a PSM generated through the solEngine. Each object within the PIM model is translated by the solEngine to an instance of the Ethereum model using mappings. From a high-level perspective first main element such as agents, assets and smart contracts are mapped to matching Ethereum classes. Dedicated functions then form instances that are included in the model. Assets require their own smart contracts that are generated using the *Asset2solAsset* function. The dedicated SC2SC function contains an algorithm to further generate elements like events, functions and modifiers. The algorithm to generate a smart contract model proceeds in the order depicted in the activity model of Figure 6.9.

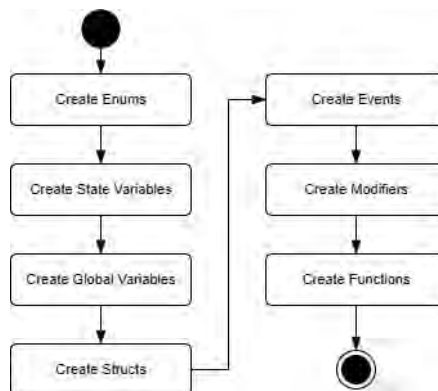


Figure 6.9: Activities of algorithm to create Ethereum smart contracts

All element handled by the SC2SC function are registered and attached to the smart contract. Functions are important elements within smart contracts and contain several local elements. During the generation of a function a copy of the function is dynamically populated to ensure that local elements are only used locally. Listing 6.4 portrays the function and the required functions in the module to transform a function in the platform agnostic model to one in the Ethereum model.

```

1
2  const thisModule = {
3      setInteractionType(interactionType){
4
5          let interaction : InteractionType
6          switch(interactionType){
7              case "None":
8                  interaction = InteractionType.NONE
9                  break;
10             case "Invokable":
11                 interaction = InteractionType.INVOKABLE
12                 break;
13             case "Transfers":
14                 interaction = InteractionType.PAYABLE
15                 break;
16         }
17         return interaction
18     }
19
20     setVisibilityType() {
21         return FunctionVisibility.PUBLIC
22     }
23     createModifier(PreConditions) {
24         //set preconditions
25         var Sequence : Array<ModifierElement> = [];
26         PreConditions.forEach(element => {
27             let newSolElement: solModifier = new solModifier(element.
28                 PreCondition);
29             let localVarEl: ModifierElement = { Modifier: newSolElement }
30             Sequence.push(localVarEl);
31         })
32         return Sequence
33     }
34     createGlobalVariable(LocalGlobals) {
35         //set Local Global
36         var Sequence : Array<GlobalVarElement> = [];
37         LocalGlobals.forEach(element => {
38             let newSolElement: solGlobalVariable = new solGlobalVariable(
39                 element.GlobalParameter);
40             let localGlobalEl: GlobalVarElement = { GlobalVariable:
41                 newSolElement }
42             Sequence.push(localGlobalEl);
43         })
44         return Sequence
45     }
46     createLocalVariable(LocalVariables) {
47         //set Local Variable
48         var Sequence : Array<LocalVariableElement> = [];
49         LocalVariables.forEach(element => {
50             let newSolElement: solLocalVariable = new solLocalVariable(
51                 element.Variable);
52             let localVarEl: LocalVariableElement = { LocalVariable:
53                 newSolElement }
54             Sequence.push(localVarEl);
55         })
56         return Sequence
57     }
58     createStruct(LocalConcepts) {
59         //set Local Structs
60         var Sequence : Array<StructElement> = [];
61         LocalConcepts.forEach(element => {
62             let newSolElement: solStruct = new solStruct(element.Concept);
63             let structEl: StructElement = { Struct: newSolElement }
64             Sequence.push(structEl);
65         })
66     }
67 }

```

```

60     })
61     return Sequence
62 }
63 createConditionalStatement(Contains) {
64     //set Conditional Statement
65     var Sequence : Array<ConditionalStatementElement> = [];
66     Contains.forEach(element => {
67         let newSolElement: solConditionalStatement = new
68             solConditionalStatement(element.ConditionalStatement);
69         let conditionEl: ConditionalStatementElement = {
70             ConditionalStatement: newSolElement }
71         Sequence.push(conditionEl);
72     })
73     return Sequence
74 }
75 createState(Execute) {
76     //set Statement
77     var Sequence : Array<StatementElement> = [];
78     Execute.forEach(element => {
79         let newSolElement: solStatement = new solStatement(element.
80             Statement);
81         let statementEl: StatementElement = { Statement: newSolElement
82             }
83         Sequence.push(statementEl);
84     })
85     return Sequence
86 }
87 createTransfer(Can) {
88     //set Transfer
89     var Sequence : Array<TransferElement> = [];
90     Can.forEach(element => {
91         let newSolElement: solTransfer = new solTransfer(element.
92             Transfer);
93         let transferEl: TransferElement = { Transfer: newSolElement }
94         Sequence.push(transferEl);
95     })
96     return Sequence
97 }
98 createReturnStatement(EndedBy) {
99     //set Return Statement
100     var Sequence : Array<ReturnStatementElement> = [];
101     EndedBy.forEach(element => {
102         let newSolElement: solReturnStatement = new solReturnStatement
103             (element.ReturnStatement);
104         let returnStatementEl: ReturnStatementElement = {
105             ReturnStatement: newSolElement }
106         Sequence.push(returnStatementEl);
107     })
108     return Sequence
109 }
110 }
111
112 class solFunction {
113
114     constructor(fu) {
115         this.functionName = fu.funcName
116         this.return = fu.return
117         this.interaction = thisModule.setInteractionType(fu.interaction)
118         this.argument = fu.argument
119         this.argumentType = fu.argumentType
120         this.visibility = thisModule.setVisibilityType()
121
122         //Set the contents of the function
123         this.Modifiers = thisModule.createModifier(fu.PreConditions)
124         this.LocalGlobals = thisModule.createGlobalVariable(fu.
125             LocalGlobals)
126         this.LocalVariables = thisModule.createLocalVariable(fu.
127             LocalVariables)
128         this.LocalStructs = thisModule.createStruct(fu.LocalConcepts)
129         this.Contains = thisModule.createConditionalStatement(fu.Contains)
130         this.Can = thisModule.createTransfer(fu.Can)
131         this.Execute = thisModule.createStatement(fu.Execute)

```

```

123         this.EndedBy = thisModule.createReturnStatement(fu.EndedBy)
124     }
125 }

```

Listing 6.4: Transformation to Ethereum function

Upon completion of the platform specific model without errors the solidity code generation commences. This task is executed by the `solCodeGenerator`. The `solCodeGenerator` writes a string for each asset, agent and smart contract. In the exact same order as depicted in Figure 6.9 thereafter the other elements are written as string. Events are written using the generator (function) depicted in Listing 6.5.

```

1  class solCodeGenerator {
2
3      generateSolEvent(_eventName: string, argument: Array<string>,
4                      argumentType: Array<string>) {
5          var eventString : string
6          eventString = "\tevent " + _eventName;
7
8          //@dev: to match the arguments and argumentTypes
9          let matchedArgumentandTypes = zip(argument, argumentType);
10
11          var argCount = 1;
12          eventString = eventString + "("
13
14          //@dev: to append each paired argument and type to the event code.
15          for(let matchedArgs of matchedArgumentandTypes){
16              if(argCount < matchedArgs.length){
17                  eventString = " " + eventString + matchedArgs[0] + " " +
18                      matchedArgs[1] + ", ";
19              }
20              else{
21                  eventString = eventString + matchedArgs[0] + matchedArgs
22                      [1] + " ";
23                  break
24              }
25              argCount++
26          }
27      }
28  }

```

Listing 6.5: Solidity event code generation

The combined strings of all elements constitute to for instance a smart contract. Any results of this step can thereafter be implemented in the Solidity Remix editor that can compile the high level code to EVM instructions.

6.6.3 Hyperledger Fabric PSM Creation and Code Generation

For Hyperledger fabric the tool enables for the creation of JavaScript based smart contracts. First a PSM is created using the `hyperledgerEngine`. Like for the Ethereum platform, each element in the PIM is mapped to one or more elements in the PSM. With the distinction however, that the elements are stored in separate files. As a first step a meta element is created that contains the ACL, Query and CTO file. These files, or rather the models for these files are transformed in this order. The CTO file containing Events, Participants and so on functions as a register for the possibilities in the smart contract. The `hyperledgerEngine` first transforms the Enums in the PIM to Enums suitable for Hyperledger. Thereafter the concepts in the PIM are transformed into concepts for the PSM. Following, the algorithm proceeds in the order depicted in Figure 6.10. Relationships stipulate an association between two elements within the model and during the execution it is checked if the associated elements exist.

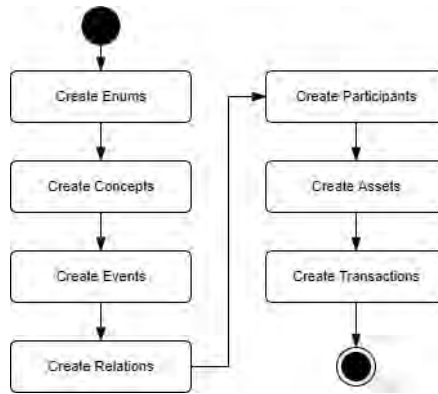


Figure 6.10: Activities of algorithm to create Hyperledger smart contracts

When the algorithm has finished with the elements related to the model needed to create the PSM it will commence to write the logic within the smart contracts themselves. First all typical properties like name and whether the contract is abstract are set. The `nameSpace` attribute from the Meta class is used here to set the `nameSpace` property to capture where the smart contract is deployed. This is important in the translation is without this property the smart contracts would not be able to connect to the network. Thereafter the program proceeds to creates the states of the smart contract and simultaneously the related, and required queries. Listing 6.6 shows the code used for the tool to transform a platform agnostic smart contract into a smart contract that can be deployed on a Hyperledger Fabric blockchain.

```

1  const thisModule = {
2
3    createState(Functions) {
4      //set Statement
5      var Sequence: Array<hyperFunctionElement> = [];
6      Functions.forEach(element => {
7        let newSolElement: hyperFunction = new hyperFunction(element.
          Statement);
8        let statementEl: hyperFunctionElement = { Function:
          newSolElement }
9        Sequence.push(statementEl);
10     })
11     return Sequence
12   },
13
14   createState(GlobalVariables) {
15     //set Statement
16     var Sequence: Array<hyperVariableElement> = [];
17     GlobalVariables.forEach(element => {
18       let newSolElement: hyperVariable = new hyperVariable(element.
          Statement);
19       let statementEl: hyperVariableElement = { Variable :
          newSolElement }
20       Sequence.push(statementEl);
21     })
22     return Sequence
23   }
24 }
25
26
27 class hyperSmartContract {
28
29   constructor(sc) {
30     var imp: Array<hyperFunctionElement> = sc.Import.forEach(el => {
31       let collect = []

```

```

32         el.Import.Imports.forEach(lib => {
33             collect.push(lib.Library.Functions)
34         });
35     });
36
37     var afu = this.createFunction(imp);
38
39     this.SCName = sc.SmartContractName,
40     this.abstract = sc.abstract,
41     this.nameSpace = thisModule.nameSpace,
42     this.States = thisModule.createState(sc.GlobalVariables),
43     this.Functions = thisModule.createFunction(sc.Functions).push(
44         imp)
45     }
46 }

```

Listing 6.6: Hyperledger Fabric smart contract transformation

Next, functions are created with coherent elements like parameters and decorators. Contrary to Ethereum smart contracts, Hyperledger Fabric smart contracts cannot use the concept of a modifier. Only rules are available that control the access to a function. Rules are first created to govern who can call the functions, and when from the set of preconditions related to the function in the PIM. To cater for the translation of the other PreCondition types like "temporal", "evaluative", and "causal" an equivalent solution needs to be developed. Any PreCondition with the aforementioned type will first be filtered out of the complete set of preconditions and thereafter translated into a conditional statement. From this point onward the generator follows an almost identical as the algorithm used for creating an Ethereum smart contract model.

6.7 Discussion

The models presented in this chapter delineates the blockchain architecture of two of the most popular blockchain platforms: Ethereum, a public blockchain and Hyperledger Fabric, a framework for private blockchains *in the context* of smart contracts. By investigating the architecture of these distinct platform especially in the context of smart contracts we lay bare the relation among smart contract concepts and those on the blockchain. Some other works [274, 276] also investigated this relation yet not in the breadth and depth as we have. Other research tends to sketch abstract perspectives on smart contracts while we sought to make implementable models that are employed for the specific purpose of MDA. The presentation of these models further advances our understanding of the two most popular blockchain platforms. In doing so we not only discussed the relation between blockchain and smart contracts, but in addition the concepts used for the smart contracts themselves.

Perhaps the most important contribution of this work are not the PSMs, but more the analysis that resulted in the creation of the PIM. We presented models for the two most dominant blockchain platforms, and demonstrated that their design principles (private vs. public) can be reconciled into a PIM. At the moment most DApps are developed for, and deployed on the Ethereum blockchain or a Hyperledger Fabric network. However, myriad of blockchain platforms exist that cater for the development of smart contracts. Further investigation into the architecture and smart contracts of these platforms is needed to identify more commonalities and whether these fit the models.

In constructing our PSMs and PIM we employed a Delphi method with a bottom-up approach with relevant literature as support. Although these models were built following a structured approach the number of smart contract developers that participated in the Delphi study (11

participants) is limited. This could potentially pose a threat to the validity of the construction of the model. Given the maturity and rapid developments in the field of BCT and smart contracts there is not a large pool of senior developers available however, which is an impediment to the further development of BCT in general [247]. Although the relationships between the concepts have been verified and validated by experts, the attributes and data types have not.

From our analysis in Section 6.5.1 it seems that Blockchain and MDA seem to be a good fit. The nature of BCT inherently restricts some options of the options when programming a smart contract. In contrast, when programming regular applications programmers are not confined by the rules a blockchain infrastructure imposes on the design of the application. This research pointed out some interesting commonalities and differences between the platforms. For instance, the fact that Ethereum has one unified blockchain allows any blockchain to retrieve information on its state while this requires somewhat more complex operations for a Hyperledger chain.

Another important difference is how the platforms arrange their preconditions. In addition to the three patterns identified by Ladleif and Weske [150] we argue that a fourth is observable: the access control pattern. We therefore included this pattern in the model and specified how they are modeled. Albeit that OpenZepelin suggest offers off-the-shelf code for all of these patterns, users are free to define their own preconditions. Hyperledger has only the Access Control pattern (Rule) as a precondition type. However, the concept of a Rule on a Hyperledger blockchain caters for fine-grained specification of the operations that conflict with the principles of a public blockchain such as transparency and immutability. This problem was resolved in this work by stipulating constraints for when the targeted platform is Ethereum, and dis-allowing the option to confine read and delete operations. Given the distinct principles of these blockchains we argue that there was little leeway for other options.

Both platforms support the notion of For and While iterations (loops) over data. Iterations are not supported within the current platform agnostic and specific models. Further studies that take these notions into account, will need to be undertaken. Another concern is the verbosity that PIM model presented here introduces, as it is difficult to incorporate the use of "syntactic sugar" into the metamodel. For example, a global variable in Ethereum could simply be declared using the statement `msg.sender`. Instead in our PIM the global variable would first need to be declared and only thereafter operations could be performed with it. While this approach would only increase execution time when using such declarations for Hyperledger, for Ethereum it would incur unnecessary costs. On the Ethereum platform users pay for each executed operation and thus more operations amount to higher costs.

One of the advantages of MDA is that it allows for the stipulation of constraints that can be employed specify what associations a class might have or what values its attributes might contain. Testing smart contract remains a pervasive issue that remains to be solved and limited tools for development are available [253]. Models that are presented in this chapter are usable for MDA and therefore aid in the latter. The former is only addressed in a limited fashion. Constraints for the models avoid some problems like function overloading and the performance of operations on non-declared variables. Future research endeavors could investigate how the combination of MDA and other testing tools could further diminish potential bugs of the semantic kind.

Finally, all information regarding the PIM is administered to the tool in a JSON format. We explain in Section 6.6.1 how the file is thereafter validated in terms of the correctness of the PIM. However, a more user friendly input interface might be desirable when the models

are to be used in practice. Besides this consideration, in Chapter 3 we discussed that smart contract are part of a blockchain system and cannot be regarded in isolation. This research discusses how to write smart contracts using MDA, it does not investigate how the smart contract could be directly connected to several other components within the blockchain platforms. An example of a such a component is a front-end that serves as an interface between the smart contract and the user.

6.8 Conclusion

The current chapter set out to develop an platform independent model from the perspectives of the two most prominent blockchain platforms: Ethereum and Hyperledger Fabric. Metamodels encompass several elements that combined constitute to a model. Following a collaborative method we developed the metamodels for these platforms with the help of 11 experts. During several consecutive rounds in a Delphi study we identified all relevant concepts requires to built the metamodels. With this data we made implementable models that are employed for the specific purpose of MDA. The construction of these models further advances our understanding of the two most popular blockchain platforms. In doing so we not only discussed the relation between blockchain and smart contracts, but in addition the concepts used for the smart contracts themselves.

Thereafter the research has set out to create a PIM based on the commonalities between the concepts that both platforms use. The comparison identified a large set of commonalities however some distinctions were also identified. Among these distinctions is the difference how user interaction with the smart contracts is restricted by preconditions. Another dissimilarity between the platforms is the manner smart contract can obtain data concerning their own state(s) or that of the blockchain. By investigating the architecture of these distinct platform based on commonalities and differences especially in the context of smart contracts we lay bare the relation among smart contract concepts and those on the blockchain. The identification of these relations aid in the design of future systems that will further ease the development of smart contracts.

Perhaps the most important contribution of this work are not the PSMs, but more the analysis that resulted in the creation of the PIM. We presented metamodels for the two most dominant blockchain platforms, and demonstrated that their design principles (private vs. public) can be reconciled into a PIM. Moreover, the PIM provides the basis for a system that aids in the creation of smart contracts that can be designed in a platform agnostic manner and can thereafter be deployed on any blockchain.

Chapter 7

Discussion

7.1 Introduction

The research question this study aims to address is how can smart legal contracts be developed in a manner that constitutes to a legally binding contract, and that can enforce their terms and conditions within that contract, regardless of the blockchain platform. In this chapter we will formulate an answer to this question to enhance our understanding of what method would facilitate the development of smart legal contracts. Due to the interdisciplinary nature of smart legal contracts the development of such a method entails that prerequisites from two distinct domains are to be met. A set of six requirements at the feature level are specified in Chapter 1, Section 1.5 to make these prerequisites explicit. Based on these requirements we set out to develop an artefact in the form of a method that satisfies these requirements. The MDA philosophy is a suitable theoretical underpinning for the design of a method that enables the development of smart legal contracts. Thus, the cornerstone of the overall method proposed in this dissertation to develop smart legal contracts is the MDA philosophy. In line with this philosophy for software development, we have presented a domain ontology in Chapter 5 that enables users to create CIMs that captures requirements from the legal perspective. Chapter 6 presents a platform agnostic and specific meta model for smart contracts. The current chapter reflects on the results attained by constructing the metamodels and discusses the evaluation of these results in light of the requirements for the overall method. This evaluation points out demarcations, limitations, and opportunities for future research. The outline of this chapter will be as follows: First we will discuss the results of this research in Section 7.2 following the requirements defined in Chapter 1, Section 1.5. Next, the chapter discusses the limitations and threats to the validity of this research. Suggestions for future research are thereafter discussed in Section 7.4. Finally, in Section 7.5 conclusions are drawn on the findings of the discussion.

7.2 Design Principles and Lessons Learned

In this section we will further discuss the results of Chapter 5 and Chapter 6. One of the first findings of this research is that the creation of smart legal contracts touches upon aspect from two distinct disciplines: the legal and software engineering. Therefore, what method would cater stakeholders from both domains when developing smart legal contracts? Does the method allow for the incorporation of the domain specific knowledge stemming from both disciplines? The results of this research demonstrate that MDA provides a solid method to facilitate stakeholders from both disciplines in several ways: First, it provides a manner to capture domain knowledge and makes it explicit to all stakeholders providing and provides a lingua franca for communication between stakeholders. Second, MDA emphasizes on the

platform agnostic development of software that can be re-used for different platforms. Finally, explicit transformation rules allow for traceability throughout the translation from legal specifications to a smart contract. These advantages have led to the adoption of MDA as the philosophy underpinning the approach to develop a solution that satisfies the requirements.

7.2.1 A Unified Domain Ontology for Smart Legal Contracts

Legal professionals require a low code or no code software solution to understand how concepts in a legal contract are related to the translation to programming concepts in a smart contract. On the other hand, programmers would benefit from such a solution because it makes domain concepts explicit easing the coding process. The combination of these two considerations led us formulate the following requirement:

Requirement 1: The method needs to cater for a manner to make domain concepts explicit so that legal professionals can stipulate the requirements for their smart legal contract.

The proposed solution fully satisfies this requirement. In identifying relevant concepts, and constructing our ontology we aimed at using concepts familiar to legal practice. Informed by past efforts in the field of legal requirements engineering, multi agent systems and electronic contracting we set out to extract all relevant concepts to legal contracts. Our investigation encompasses 29 works and we are confident that the breadth and depth of our analysis ensures that most relevant concepts are identified. That is, with the exception of some concepts discussed in Section 5.9 like a condition precedent and subsequent that are mentioned in law literature. Our ontology is primarily developed as a means for legal experts to specify their smart legal contract. An interesting observation is that each of the concepts in the domain ontology is in essence a user requirement at what Gorschek and Wohlin [95] would describe as the functional level for a smart contract. Each concept in the domain model should correspond to a kind of representation in the smart contract. With the main objective of requirement 1 in mind, we argue that the domain ontology assists smart contract programmers in understanding the legal requirements that a legal expert has provided.

Albeit that our ontology enables the creation of a model that encompasses domain specific concepts, the specification of the model is stored as a JSON format. We posit that this format is difficult to read for most legal practitioners. The method we proposed in this research does not facilitate the visualization of a model created by a legal experts and would require an additional presentation layer. Ladleiff et al. have made efforts [150, 152] to visualize smart legal contracts using BPMN. It has been argued however, that such an approach would be too simplistic to model the intricacies of a legal document [34]. An alternative approach is to directly couple parts of the user specified model to words in the legal contract. Systems to develop smart legal contract the likes of OpenLaw¹, and Accord Project² provide a markup language to allow legal practitioners to manually denote specific concepts within the digitally stored version of the legal contract. However, it remains to be seen whether the markup language is understandable for a legal practitioner.

Bench-Capon and Gordon [29] argue that one of the most important requirements for representing legal knowledge is isomorphism. Isomorphism in the context of legal knowledge systems means that there is a well-defined correspondence between the source document and the representation of the information they contain used in the system. To achieve this feat would require another metamodel suitable to model the *contract document*, potentially

¹<https://www.openlaw.io/>

²www.accordproject.org

with lay-out. Some preliminary research results³ demonstrate that the model for the contract document could be easily incorporated and coupled to the model specified by the legal expert. An example of how this could be achieved can be found in Listing 7.1.

```

1  {"contractName": "ContractExample",
2  "contractId": 1,
3  "contractStructure": {
4    "pages": [ {
5      "page": {
6        "pageNumber": 1,
7        "sentences": [
8          {"sentence":
9            {"words": ["The", "rent", "of", "the", "premises", "will", "be", "$", "1000",
10              ". "],
11            "font": "Times", "nr": "1",
12            "info": {"Definiens1": [0, 1], "Definition1": [6], "Resource1": [7],
13              "Quantity1": [8]}}},
14          {"sentence":
15            {"words": ["The", "tenant", "will", "pay", "the", "rent", "in",
16              "advance", "on", "the", "1st", "day", "of", "each", "month", ". "],
17            "font": "Times", "nr": "2",
18            "info": {"Promissor1": [0, 1], "Action1": [3], "Quantity4": [4],
19              "Resource4": [5], "TimeExpression4": [10], "RelativeTimepoint1": [12],
20              "TimeUnit2": [15, 16]}}},
21          {"sentence":
22            {"words": ["The", "tenant", "must", "pay", "a", "late", "charge", "of", "$",
23              "500", "for", "each", "payment", "that", "is", "more", "than", "30",
24              "days", "late", ". "],
25            "font": "Times", "nr": "3",
26            "info": {"Promissor2": [0, 1], "Action2": [3], "Quantity6": [9],
27              "Resource5": [8]}}}
28        ]}}}
29    "contractModel": [
30      {"party": { "partyId": "Party1",
31        "partyName": "Bob Book",
32        "entityType": "natural" }},
33      {"party": { "partyId": "Party2",
34        "partyName": "John Doe",
35        "entityType": "natural" }},
36      {"definition": {
37        "definitionId": "Definition4",
38        "definition": [
39          {"definiens": {"definiensName": "the rent",
40            "definiensId": "Definiens4"}},
41          {"defiendum": [
42            {"resource": {"resourceId": "Resource1",
43              "resourceName": "rent",
44              "quantity": {
45                "quantName": "the",
46                "quantId": "Quantity1",
47                "value": "500"}}}}]}]}]}
48      {"clause": {
49        "clauseId": 1,
50        "clauseType": "Normal",
51        "TriggeringEvent": {
52          "triggerId": "Trigger1",
53          "Contains": [
54            {
55              "event": {
56                "eventName": "start lease",
57                "eventId": "Event1",
58                "eventType": "temporal"}}},
59            "Conditions": [
60              {"condition": {
61                "conditionId": "Condition1",
62                "Contains": [

```

³The author of this dissertation has conducted additional research to investigate how the structure of legal documents could be modelled.

```

60     {
61         "action": {
62             "actionId": "Action1",
63             "name": "pay the Rent",
64             "modal": "OBLIGATION",
65             "group": "Pay",
66             "repeated": false,
67             "status": "unfinished",
68             "resource": {
69                 "resourceId": "Resource4",
70                 "resourceName": "rent",
71                 "definedAs": "Definition4",
72                 "quantity": {
73                     "quantName": "the",
74                     "quantId": "Quantity4",
75                     "value": 500,
76                     "definedAs": "Definition4"
77                 }
78             },
79             "promissor": {
80                 "promissorName": "the tenant",
81                 "promissorId": "Promissor1",
82                 "party": "Party1"
83             },
84             "promissee": {
85                 "promisseeName": "the landlord",
86                 "promisseeId": "Promissee1",
87                 "party": "Party2"
88             },
89             "term": {
90                 "termId": "Term3",
91                 "type": "span",
92                 "timeexpression": {
93                     "timeExpressionId": "TimeExpression2",
94                     "type": "on"
95                 },
96                 "timePoint": {
97                     "timeSpan": {
98                         "timeSpanId": "TimeSpan2",
99                         "start": {
100                             "date": {
101                                 "day": 1,
102                                 "month": 1,
103                                 "year": 2021
104                             }
105                         },
106                         "end": {
107                             "date": {
108                                 "day": 1,
109                                 "month": 1,
110                                 "year": 2023
111                             }
112                         }
113                     }
114                 }
115             },
116             "repeatedEach": {
117                 "timeUnitId": "TimeUnit1",
118                 "type": "month",
119                 "quantity": {
120                     "quantName": "1",
121                     "quantId": "Quantity5",
122                     "value": 1
123                 }
124             }
125         }
126     },
127     "Remedies": [
128     {"remedy": {
129         "remedyId": "Remedy1",
130         "Contains": [

```

```

132 {
133   "action": {
134     "actionId": "Action2",
135     "name": "pay a late charge",
136     "modal": "OBLIGATION",
137     "group": "Pay",
138     "repeated": false,
139     "status": "unfinished",
140     "resource": {
141       "resourceId": "Resource5",
142       "resourceName": "late charge",
143       "definedAs": "Definition6",
144       "quantity": {
145         "quantName": "the",
146         "quantId": "Quantity6",
147         "value": 500,
148         "definedAs": "Definition6"
149       }
150     },
151     "promissor": {
152       "promissorName": "the tenant",
153       "promissorId": "Promissor2",
154       "party": "Party1"
155     },
156     "promissee": {
157       "promisseeName": "the landlord",
158       "promisseeId": "Promissee2",
159       "party": "Party2"
160     },
161     "term": {
162       "termId": "Term4",
163       "type": "relative",
164       "timeexpression": {
165         "timeExpressionId": "TimeExpression4",
166         "type": "after"
167       },
168       "timepoint": {
169         "relativeTimePoint": {
170           "relatTimePointId": "
171             relatTimePoint1",
172           "relatedTo": "Action1",
173           "timeUnit": {
174             "timeUnitId": "timeUnit2",
175             "type": "day",
176             "quantity": {
177               "quantName": "30",
178               "quantId": "Quantity7",
179               "value": 30}}}}}}}}}}
180 }

```

Listing 7.1: Representing a contracts' structure to achieve isomorphism

In the example shown in Listing 7.1 the file for the contract encompasses two parts: (1) `contractStructure` and (2) `contractModel` where the former denotes the direct coupling between words in the contract and the concepts within the user defined model, and the latter the user defined model itself. The contract structure is defined in line 7 to 24 of the listing. As can be seen on these lines, sentences included in the contract structure have a `words` and `info` attribute. In the `info` set of each sentence concepts are linked to words by attaching the index number of the word(s) (in brackets) to the concepts `Id` (e.g., `Action1`). Further information about other concepts related to `Action1` can be found on line 62.

7.2.2 Representation of legal contracts

Translating legal contracts into smart contracts is no trivial task. How arduous this task actually is becomes apparent from the results presented in Chapter 5 Section 5.5 where 21

problems are fleshed out that hamper the adoption of smart legal contracts. An interesting finding is that when closely examining the concepts proposed in literature we observe that problems are related to specific parts of a smart legal contracts' life-cycle: There are problems that are related to the drafting and coding and other problem that are related to monitoring, execution and enforcement. This finding resonates with those of Beydoun et al. [31] that in their work on MAS discern between designtime and runtime concepts. Drummer and Neumann [74] do not make this distinction but rather map these problems to a legal and regulatory level, contract level, technical implementation level. None of the problems they identify at the technical implementation level such as scalability, privacy and interoperability are included in our 21 problems. We posit that these problems are not uniquely related to smart legal contracts but as discussed in Chapter 2, to BCT in general. Being aware of the problems related to different parts of the life-cycle, the method devised for this research had to satisfy the following requirement:

Requirement 2: The method needs to cater for the specification, and potential resolution of issues when coding and drafting.

By fleshing out the problems and relating these to concepts in the domain ontology our method partially fulfills this requirement. We found that what can be perceived vagueness in a contract is not always unintentional but rather open texture often employed to create leeway for unforeseen future situations. However, it is exactly these unforeseen situations that can create problems when translating them to smart contracts. Smart contracts can only compute on exact concepts like numbers. Although we did not provide a direct solution to solve this part of the problem the ontology offers a manner to denote open texture so that it may become explicit. On the other hand, because the problem is explicit any future state that the smart legal contract would encounter during it's execution would be known. That is, if there is a manner to inform a smart legal contract what a promisee would understand under that specific circumstance as "reasonable" and "promptly".

Another problem we identified was the use of modalities in contracts. The results of our study suggest that modalities are always related to an action that one of the parties has to perform. In their work Ladleif and Weske [149] expound on how to model obligations and permissions using BPMN choreographies. The notion of legal states is introduced to denote which activities have been performed. Similarly, our ontology encompasses action with their own unique status. Modalities like permissioned actions are problematic because the smart contract does not know in advance what choice the party that is permitted wants and is going to make. Ladleif and Weske [149] indeed suggest that this choice should be made beforehand by the party who is granted the permission. They suggest the use of events to notify a party that they are allowed to make a choice. How the choice is then conveyed to the smart contract remains unclear however. Despite it's invaluable contribution their study does not include a choreography for prohibitions that we besides obligations and permissions identified as important modalities.

An interesting observation is that a solution to solve the problem of permissions seems to require the same design characteristics as one that resolves vagueness: (1) The smart contract knows that a party must be consulted. (2) The appropriate party should be notified that they are allowed to make a choice what action to take or to determine what they find reasonable. (3) The appropriate party makes a choice and communicates this choice to the smart contract. Thereafter the smart contract would be able to act upon the choice made by the party. The findings from our research suggest that ambiguity, in the form of coordinating conjunctions in contracts, is another problem that is hampering the translation of a legal contract to a smart contract. We tackle this problem first by specifying in the ontology that attached to each action, there is one promisor, promisee, and resource. The notion

of exclusivity (or coordination conjunction) is catered for by adding an attribute exclusive to. Again, we can detect a pattern here similar to that for permitted actions the smart contract here needs to be aware which of the two actions has been performed.

While the domain ontology offers a notation to make most problems explicit there are two problems that we did not address: Firstly, we did not address the problem of understandability directly but provided a domain ontology constructed with concepts we argue are native to legal professionals. As argued in the prior section the ontology can be extended to also serve as the foundation for a more direct coupling between the legal contract and the specification. Secondly, resolving the issue of proving the existence of a contract is not discussed in this work. For our method we assume that a written version of the legal contract in question exists. Ideally the legal contract would be stored simultaneously with the smart contract upon deployment. Naturally, a smart legal contract is stored on a blockchain however, it is beyond the scope of this research to investigate how this storage could be facilitated for the legal contract. However, we posit that this is an important problem for two reasons namely, given that the legal contract serves as the specification document for the smart contract each line of code should be traceable to the original document. Most important it would provide the contracting parties with a readable format of the agreement. Prior work has investigated how to store files using a BCT based system and might provide a solution for this problem. A study by Magrahi et al. [168] presents a system to store and share files off-chain while guaranteeing the existence of said files using the blockchain. An on-chain approach to store files is suggested in the work by Huang, Chang, and Wu [116]. Both approaches could potentially be integrated in the method.

7.2.3 Enforcement with smart legal contracts

An important finding of this research is a second set of problems are related to the parts of a smart legal contracts' life-cycle throughout when the smart contract is deployed, monitors and enforces the contract and when disputes arise. This second set of problems coincides with another set of requirements. This led us to define the following feature requirement:

Requirement 3: The method needs to cater for the specification, and potential resolution of issues related to enforcement of terms and conditions by a smart contract.

By constructing the domain ontology we provide a means to foresee the problems that might arise during the deployment or execution and monitoring of the contract. A comparison of the findings with those of Drummer and Neumann [74] shows that they identify three identical issues at the legal and regulatory level. Among these problems is the problem that to which the authors refer to as "Unclear domiciling and jurisdiction" this resonates with the two problems we have labelled as the Applicable Law and Jurisdiction problem. Another identical finding is that a smart contract itself and any transaction a smart legal contract has produced cannot be reversed. We observe that consequently this causes a potential complication: Similar to Drummer and Neumann we found that this leads to the problem that when disputes arise. Parties might resort to litigation or arbitration to settle their dispute resulting in a court order or agreement to undo parts of the contract. In the same vein, Drummer and Neumann discuss this problem in the context of whenever a violation of consumer rights occurs and the court orders the contract to be declared void yet it cannot be reversed. Globally consumers are protected by several consumer rights that safeguard fair trading practices. The aforementioned authors found that in such cases there are no remedies for consumers when their rights are being violated or illegal agreements are made. Closely related to their identified problem, we have referred to as illegal contracts where clauses in a contract might contradict existing laws making them illegal.

Taken together, a broad observation is that currently in most countries much remains unclear about how smart legal contracts should be regarded in light of current legislation [106]. Furthermore, even in spite of legislation for smart contracts this might not directly result in smart contracts adhering to these standards as the jurisdiction and applicable laws might not be apparent and are thus not incorporated. Taking into consideration these observations and combining them with the insights attained from the discussion in the prior section on the issue of references in contracts we argue that these problems compound one another. Assuming that the smart legal contract has a legal contract equivalent there needs to be a reference to these laws as a smart contract most likely does not operate in a legal vacuum. These laws are not available in digital form and thus referring to said laws is not possible. Another consequence of the current lack of a legal framework for smart contracts and blockchain in general is that both technologies are prone to criminal abuse. Especially given the pseudo-anonymity that a blockchain provides that prevents criminal investigation. Drummer and Neumann mention this as a specific issue that hampers the adoption of smart contracts for legal contracting practice because parties engaging in contracting via a smart contract could theoretically not know each other.

One of the most prospects of smart contracts is their potential to monitor and execute transactions autonomously. Moreover, based upon *a priori* defined conditions a smart contract is able to self-enforce conditions. However, as discussed in Chapter 5, section 5.5 the characteristics of a smart contract will raise several issues when tasked with monitoring, executing and enforcing legal contracts. Smart contracts tasked with monitoring a legal contract require information on the activities actors promised to or external data stemming from the 'real world' outside of the self-contained blockchain system. Through their research Drummer and Neumann similar to ours, identify the need to align data that the smart contract requires to the 'real world' and how this can become problematic.

Because in our ontology actions have their own unique status information concerning the current state of affairs on promises of performance can be captured. A problem posited in literature is that initially smart contracts have been designed to predominantly to monitor and execution of payments, legal contracts might contain a wider range of action types. An observation therefore is that not only should the smart contract be able to monitor that an action has been performed but whether it was the type of action specified in the legal contract. As a remedy the ontology includes the possibility to specify the type of an action. In this manner users can make a distinction between different types of actions. For each action type specific patterns could be devised in line with the desired semantics of the action.

When monitoring conditions by verifying facts with regard to the fulfillment of promises or other facts smart contracts require access to information or data outside of their self-contained system. Ladleif and Weske [150] notice that several types of data sources can be discerned that a smart contract can consult to attain this information. In Chapter 3 we have discussed the pivotal role of oracles in providing smart contracts with the data they require to establish such facts. Further investigation into oracles by Ladleif, Weber, and Weske [148] suggests that several oracle-based implementation strategies are required as there are different types of oracles. In their study storage and request-response oracles types are discussed. Storage oracles store their data in a smart contract and this data is updated regularly, while request-response oracles are implemented using a smart contract that returns information from another blockchain upon request of a smart contract. An important ramification of this design is however, that smart contracts only possess data in hindsight after it has called the oracle smart contract.

Another important observation from the discussion in Chapter 5 is that several types of assets exist. When further dissecting the digital representation problem we identified in Chapter 5, Section 5.5 we noticed that there are some parallels between the 'wide range of actions' problem and this problem. When executing monitoring itself or executing transfers from one agent to another the smart contract needs to be knowledgeable about what kind of asset is involved in the exchange, similar to action types. The ontology presented in Chapter 5 caters for the specification of the asset type yet we did not provide a complete set of all asset types. Research conducted by Zhu et al. [286] provides an interesting classification of assets that encompasses digital currency assets, data assets, physical assets and intangible assets. The former are Ether, Bitcoin and other cryptocurrencies we have referred to as "native assets" in Chapter 6. Data assets concern data related to companies, persons and algorithms these assets have the commonalities that they are data products. Physical assets have a physical substance and are related to things like house, cars, paintings etc. whereas intangible assets do not. Examples of such assets are intellectual property, goodwill and brand recognition. We discussed in Chapter 6, Section 6.5.1 that currently fungible tokens and NFTs are predominantly used to represent assets. The work by Zhu et al. [286] does not discuss how these asset types are related to theirs, with the exception of the digital currency assets. Taken together, it seems that the last three categories of assets posited by Zhu et al. [286] seem to be in line with the use of NFTs. Furthermore, we observe that akin to actions, further investigation is required to classify and standardize assets on the blockchain.

Smart contracts rigidly enforce the terms and conditions contained in their logic. However, in some cases this characteristic is not desirable. Fostering relationships is an important aspect of the purpose of a legal contract [179]. Maintaining a cordial and harmonious relationship can outweigh enforcing sanctions and administering remedies. Contracting parties can expect no flexibility from a smart contract as at the moment a mechanism to prevent the automatic enforcement of its terms are absent. On the other hand, the introduction of said mechanisms would again scarify the automatizing properties of the smart contract. A solution that might be viable is the inclusion of an emergency break that terminates the enforcement. Such a mechanism would already be required in situations of Force de Majeure where parties are discharged from their obligations. Tjong Tjin Tai [249] discusses several solutions to solve this issue. Our ontology does not provide a mechanism to solve the problem directly yet we argue that any legal contract modelled should contain such a clause to cater for these events.

The non-performance of a party can also lead to disputes for which the law provides several remedies. One solution would be to amend the legal contract, in tandem the smart contract would also be required to be adjusted. However, since smart contracts are immutable this would not be possible. Another solution would be that parties engage in arbitration. A problem identified in the study by Drummer and Neumann [74] and not by ours is that hitherto a mechanism that facilitates arbitration between parties is absent. Moreover, they consort like the problem we identified as undoing smart contracts is that litigation might order the undoing of the contract as if it had never existed in the first place. Marino and Juels [170] discuss how to develop standards for altering and undoing smart contracts. More important, they flesh out the conditions under which the smart contract is ought to be amended and undone. The work provides invaluable contributions to a solution for the amendment and undoing problem identified through this research. It is important to note that the solutions proposed in the work are based on Ethereum smart contracts. Thus it remains unclear how smart contracts based on other blockchains can implement these solutions. By presenting a platform independent meta model for smart contracts in this dissertation these patterns can be translated to multiple platforms.

7.2.4 Achieving Complete Traceability

The final requirement we have formulated is that the traceability between the legal specification (i.o. the domain ontology) and the smart legal contract created using the method is explicit. An omission of this requirement would mean that contracting parties again have to trust third parties that their contract is translated in line with what they have specified. In line with this reasoning the following requirement is specified:

Requirement 4: The method needs to cater for traceability between the specification of the legal contract to the smart contract that will embody this specification.

MDA supports traceability as the philosophy provides a record of the transformation that specifies how elements in one model have been translated to another model. In addition, if made explicit transformation rules further describe the transformation process. However, the solution partially satisfies this requirement to the extent that the transformation rules that enable the transformation between the platform agnostic meta model and the platform specific models for Ethereum and Hyperledger are explicit (see Section 6.5.3). A declaration of these transformation rules would allow anyone willing to trace how elements in the platform agnostic metamodel are translated. A set of transformation rules that this work does not discuss is the transformation from concepts used in the domain ontology to those in the platform independent metamodel for smart contracts.

An adequate transformation from the concepts in the domain ontology to a smart contract PIM requires a fine-grained specification of the transformation rules between these two metamodels. Although many of these transformation rules seem intuitively obvious, further research is needed to determine how specific concepts should be represented in a smart contract. An important aspect to consider when carrying out this research is to consult legal professionals and inquire whether the representation of the concepts in the domain ontology is accurately reflected as smart contract concepts. Moreover, we observe that each problem discussed in Chapter 5, section 5.5 demands a solution that entails that a novel mechanism or routine is developed. Although some of these mechanisms or routines have been suggested [149, 170, 249] many open issues remain.

7.2.5 Development of Platform Agnostic Smart Contracts

The cornerstone of the MDA philosophy to develop software is the use of several models portraying several abstraction layers. One of the benefits of using MDA for the development of software is the potential to re-use code across platforms. In Chapter 1, Section 1.5.2 we discussed that one of the problems of current approaches to write smart legal contract is that they are not platform agnostic. Consequently, this dissertation set out to develop a method in line with the following requirement:

Requirement 5: The method to needs to cater for the creation of smart contracts that are platform agnostic to facilitate cross platform usage.

The construction of these meta models led to several findings that we will now further discuss. Firstly, the distinction between private and public blockchains lies in the limitations that they impose on participants part of the blockchain not directly related as parties in a smart contract. Both Hyperledger Fabric and Ethereum smart contracts include the notion of pre-conditions. Whereas other work [150] identifies three types of pre-conditions we notice a fourth in the form of access control patterns. Another observation is that other research in software engineering has introduced the notion of Role Based Access Control (RBAC). The distinction between access control and RBAC is AC patterns stipulate rules for an operations that are directly coupled to an actor whereas when using RBAC patterns the

operations is tied to a role [221]. The Hyperledger Fabric framework already caters for RBAC patterns as a participant's wallet can contain multiple identities for several roles. On the Ethereum blockchain these mechanisms are not explicitly available although our literature review in Chapter 2 shows that some work by Cruz, Kaji, and Yanai [63] has aimed to resolve this problem.

Secondly, this research finds that due to its public nature of Ethereum does not allow users of smart contracts to limit the readability of their transactions or smart contracts. Private blockchains like Hyperledger Fabric have been designed with this feature in mind [274]. Paradoxically, the findings from Chapter 2 suggest that public blockchains acknowledge these privacy concerns and have started to adopt methods to provide similar features. A clear example is the utilization of ZK-SNARKS to obfuscate transaction data while making it verifiable to miners. In [135] this method is further extended with a mechanism called HAWK that allows for the obfuscation of code or state variables that should not be publicly visible. The findings of our research suggest that the inclusion of a pattern to govern the readability of transactions and potentially smart contracts themselves would be an invaluable addition for public blockchains as of now this specification is not available for Ethereum. Likewise, Drummer and Neumann [74] found that the absence of such options hampers the adoption of smart contracts.

Ultimately the goal of creating a platform agnostic model of the smart contract is to transform it into a platform specific model, and subsequently generate code using this model. The transformation requires a platform specific metamodel for each targeted platform that led to the following requirement:

Requirement 6: The method needs to cater for the creation of smart contracts on a specific platform.

In order to satisfy this requirement a platform specific metamodel for Ethereum and Hyperledger Fabric smart contract is presented in Chapter 6. An important consideration that this research lays bare is that the creation of the platform specific metamodels requires significant expertise and time. At the time of writing a multitude of blockchain platforms exist that support smart contracts. A practical consideration is whether creating a metamodel for each of these platforms is worthwhile. In some cases the time and costs associated with the construction of the metamodel outweigh the gains of doing so. Moreover, although there are several smart contract supporting platforms at the moment, in the future only a small selection of these platforms might remain in existence.

7.3 Limitations and Threats to Validity

Like any other study this research has limitations that will be discussed here. Employing the guidelines by Wohlin et al. [269] again, the threats to the validity for this study were identified and are discussed in this section.

7.3.1 Internal Validity

First, in constructing the domain ontology presented in Chapter 5 we reviewed literature in the field of legal requirements engineering, e-contracts and MAS. While we are confident that the number of works and the depth of our investigation covers most relevant concepts for legal contracts the search to identify studies is not conducted in a systematic fashion. Because of this method other important studies could potentially be overlooked, and thus not included in the selected works. Consequently, some of the concepts presented in such

works would not be included in this research. Likewise, studies that discussed the problems presented in Chapter 5, Section 5.5 were not identified using a systematic search strategy. Another option is to carry out the searches following the approach presented in Chapter 2.

In attaining the concepts deemed relevant for the construction of the domain ontology we employed a qualitative content analysis. An identical method has been used to flesh out the problems related to smart legal contracts. Although qualitative content analysis provides a systematic method to analyze data, there are some factors that could have influenced the outcomes of the analysis. When employing a qualitative analysis method the unit of analysis is an important factor influencing the observations [137]. For the construction of the domain ontology and when extracting problems a paragraph is adopted as the unit of analysis. It is recommended that researchers use the smallest unit of analysis possible to get the most detailed analysis [137]. However, for this research we posit that paragraphs were the most appropriate unit of analysis given that in most works entire paragraphs were dedicated to explaining one concept. Thus, focusing on individual sentences might not yield the same results.

The second phase of the data analysis process to identify the problems related to smart legal contracts and relevant concepts included open coding. Open coding is an inductive process that requires interpretation from the researcher with regard to the object of study. Any researcher conducting such an analysis is prone to several bias that could influence the interpretation of the data and thus the results. The same line of reasoning applies to the revision of the initial codes and the creation of the categories. However, the comparison in the prior section of our work to that of Drummer and Neumann demonstrated that several identical problem were identified, albeit with a different category names. A potential threat to validity would be whether all relevant concepts were included in the model. Further verification is needed to examine whether this is the case.

7.3.2 Construct Validity

Second, to construct the platform specific meta models for smart contracts there was a cooperation with several blockchain experts. An initial list of concepts was derived from existing literature on the subject. The fields of BCT and blockchain smart contracts are still in their infancy as the review in Chapter 2 demonstrates. An inherent consequence of this status might be that not all blockchains are currently discussed in literature.

Questionnaires have been administered to the blockchain experts to gauge whether all concepts required for these meta models were identified, to identify aliases and to verify whether the concepts are familiar. Although metrics like clarity, comprehensiveness, correctness, consistency and conciseness of the concepts used for the models have been assessed using different questions, the relationships between these concepts has not. Another approach is to like Holsapple and Joshi [114] present the models to the panelists.

In Chapter 5, the domain ontology is evaluated using a lease contract as a motivational example. Furthermore, the chapter also discusses the problems that arise when using smart legal contracts. However, this research did not investigate whether factors like the type of contract or other contingencies influence the plausibility of employing smart legal contracts.

7.3.3 External validity

Third, there are some threats to the external validity of this study. The platform independent meta model presented in Chapter 6, Section 6.5.2 is based on the two most prominent blockchain platforms Ethereum and Hyperledger Fabric. However, as discussed in Chapter

2 nowadays a myriad of blockchain platforms exist that cater for smart contracts. Given that the platform agnostic meta model presented in this research is constructed using the concepts of these two platforms there might be concepts native to other platforms that cannot be modeled using the current meta model. Further expansion and refinement of the platform independent metamodel based on research that includes more platforms is warranted. Besides this consideration there are some limitations inherent to the use of the Delphi method for content evaluation. A Delphi method is suitable to explore novel phenomena with the help of a discrete group of experts. The method should be perceived as inductive [69]. Therefore, further deductive research among a broader group of experts is appropriate to gathered to strengthen the empirical evidence.

The domain ontology that is presented in Chapter 5 is constructed using concepts that are found in legal requirements engineering, e-contracts, and multi-agent systems literature. However, something that remains to be tested is if the domain ontology is complete enough to specify legal contracts in all legal traditions. For instance, does the ontology covers all concepts related to legal contracts in civil law versus common law. A more broad investigation would be required to establish whether this holds true. It can be regarded as a limitation of this study that the domain ontology has not been evaluated content wise. Similar to the platform agnostic smart contract metamodel, a future evaluation of the domain ontology is an important second step towards the refinement of the domain ontology. Such an effort would require a group of legal experts with a diverse background that practice law in several legal traditions. Given the time-scope for this research assembling a group of said legal experts was deemed not feasible.

7.4 A Research Agenda for Smart Legal Contracts

The research conducted for this dissertation answers some important questions about the potential of smart legal contracts to constitute to a legally binding and enforceable legal contract. Despite the fact that this research demonstrates that smart legal contracts have this potential, like all studies it is not without limitations. These limitations are interesting venues for future research. Besides answering questions, in tandem this research raises some novel questions that future research can investigate. This section points out several opportunities for future research from a legal, technical, and business perspective.

7.4.1 Legal Research Opportunities

We hope that the ontology presented in this dissertation is the starting point of a discussion between legal scholars and smart contract developers. To further validate whether the concepts used for our domain ontology are rooted in practice, legal experts should be engaged in the further development of the domain ontology. As far as we are aware of none of the related works reviewed in Chapter 5 has involved domain experts in the creation of their models. Another interesting question that future studies can address is whether there is a difference between the concepts used in legal contracts based on the civil versus common law tradition.

In the prior section it is discussed that assets, clauses, and actions are actually umbrella terms that encompass a wide variety of more specific assets, actions and clauses types. The domain ontology presented in this dissertation includes a generic class for each of these. However, the plurality of asset, clause and action types need to be fleshed out so it becomes clear what options there are and how they should be implemented. Some attempts have

been made by practitioners using NLP techniques⁴. Similar, there is no clear overview of the actions used to denote (conditional) promises to perform or how they could potentially be grouped by semantic equivalence. Future research could focus on identifying the types of clauses and promises to perform used by legal practitioners.

Another aspect this research has not touched upon is the invaluable yet equally important matter of how to ease the specification process for legal experts. Assuming that the domain ontology presented in Chapter 5 eventually encompasses all relevant concepts, legal professionals have to work with the tool presented in Section 5.8 to indicate the concepts that they are using for the smart legal contract that they are drafting. Further evaluation is required to investigate whether this tooling is sufficient to stipulate several types of legal contracts. The evaluation of the legal tool could be carried out by letting legal professionals stipulate several legal contracts (e.g., insurance contracts, buyer-supplier contracts and temporary employment contracts) using the tool. Semi-structured interviews could be conducted thereafter to ask the legal professionals their opinion about the tool. These interviews can also serve to gather data on current the current functionalities of the tool, and to gather additional requirements.

Although the tooling would allow legal professionals to define their smart legal contracts, gathering legal requirements for a software program and by extension, requirements for a contract is a time consuming and arduous process [125]. Some research [132, 237] has introduced automated approaches to gather legal requirements that might greatly increase the ease and speed of gathering requirements for smart legal contracts. However, of the related practitioners' projects or the scholarly works none provides a specification of how their models are populated.

7.4.2 Technical Research Opportunities

A limitation of this research is that in this research two blockchain platforms are discussed and platform specific meta models thereof presented. The field of BCT and smart contracts is rapidly advancing and changing. Since the introduction of Ethereum and Hyperledger several other blockchain platforms have been established. Future research could develop platform specific metamodels for blockchain platforms other than Hyperledger and Ethereum. Once the platform specific metamodels have been constructed, then an evaluation could be conducted to determine if the platform independent metamodel can be employed to abstract smart contracts for other blockchain platforms. Developers would greatly benefit from the knowledge and perspective such an inquiry would yield.

Although blockchains are considered to be a self-contained system, in practice connecting smart contracts to oracles and other blockchains is required to get important data from outside of the system. The smart contract models presented in this research did not include any concepts related to oracles, or take into account the possibility of cross-chain transactions. Further investigation is required to determine which concepts need to be included in the models to enable the retrieval of data from oracles, and cross-chain communication. Mammadzada et al. [169] provide an interesting overview and framework of blockchain oracles that could serve as the foundation for the analysis of what code patterns would be needed to integrate the different types of oracles. The works by Lo et al. [159] and Sheldon [229] could be used to assess potential vulnerabilities in the code patterns.

⁴See LawInsider <https://www.lawinsider.com/clauses> for an example.

A review by Belchior et al. [28] provides an excellent starting point for comparison between seven different cross-chain communication methods. Future research endeavours could further examine which cross-chain communication method is supported by a specific blockchain. This examination can serve to create novel concepts related to cross-chain communication and stipulate constraints to extend the smart contract models. In the same vein, other architectural options could also be further explored to create a broader systems perspective that also includes off-chain components working in interplay with on-chain contracts. While carrying out this exploration an emphasis should be placed on the integration of decision-making options with regard to which components are off and on chain.

The blockchain and smart contract models presented in this research are to be used in combination with a tool that supports and eases the process of writing smart contracts. A limitation of this research is that the user friendliness of the tool has not yet been evaluated. Future research endeavors could engage smart contract developers in the further development of the tool to evaluate the user friendliness of the tool. A manner to further evaluate the user friendliness of the tool is to invite several smart contract developers to model a set of use cases. To make a solid comparison possible, these use cases could include well-known smart contract examples like a Solidity Simple Storage⁵ contract, Open Auction smart contract⁶, and the Hyperledger format for the Non-Fungible ERC-721 standardized token⁷. After the modeling experiment semi-structured interviews could be organized with the smart contract developers to ask their opinion about the tool. The results obtained through these interviews can be employed to assess the user friendliness of the tool, and gather further user and functional requirements.

The tool itself is used to generate smart contract code for a specific platform. Novel research is required to determine how well the code works based on a specific set of criteria. Existing formal verification methods to evaluate access control to an Ethereum smart contract [225], and methods based on automated test-case generation for smart contract code [73] are in particular useful when assessing such criteria. Other works are instrumental to evaluate the performance of the code generated for Hyperledger smart contracts [81] and potential vulnerabilities [279]. Such an inquiry should also consider how well any code patterns developed to address the issues related to smart legal contracts work as a solution from a technical and legal perspective.

7.4.3 Business Research Opportunities

The discussion in Section 7.2.3 highlights that there might be several factors that determine whether a smart legal contract can be used instead, or in tandem with a legal contract. Lumineau, Wang, and Schilke [161] suggest that one of these factors could be codifiability, that is whether it is possible to structure information or knowledge into code. An important other factor could be whether the contingencies in a legal contract that are known in advance. All conditions in a smart legal contract need to be known a priori to their deployment and therefore so must all contingencies related to these conditions. Future research could further investigate when to pursue smart contracts.

Even if it is technically possible to pursue smart legal contracts there might be economic considerations to take into account when deciding to do so. As discussed in Chapter 5 in Section 5.5, there are many problems when translating legal contracts in smart contracts. Although these problems could potentially be remedied by the introduction of novel mechanisms, this

⁵<https://docs.soliditylang.org/en/v0.4.24/introduction-to-smart-contracts.html>

⁶<https://docs.soliditylang.org/en/v0.5.2/solidity-by-example.html>

⁷<https://hyperledger.github.io/firefly/tutorials/tokens/erc721.html>

would require the inclusion of additional code in the smart contract. Each execution step that the smart contract has to perform increases the cost of its execution. Moreover, even when smart contract development tools become widely available it is likely that a smart contract developer will still have to review the code. As a result, when parties employ a smart legal contract to govern their transactions, they might incur considerable costs for their execution. This raises the question whether it is worthwhile, and if so under which circumstances, to employ smart legal contracts. The transaction cost theory [268] may be of great assistance in answering these questions.

Blockchain and by extension smart contracts may be a completely novel way of organizing collaborations and to govern transactions [161]. While smart legal contracts can rigidly enforce conditions like a legal contract, fostering relationships is another important function of a legal contract (See Section 7.2.3). The question remains how well smart legal contracts may fulfill this function. Furthermore, parties may resort to relational mechanisms to complement their legal contracts. Little is known hitherto about the implications of using smart legal contracts in concert with these relational mechanisms.

7.5 Conclusion

The current research sets out to answer one main research question: How can smart legal contracts be developed in a manner that constitutes to a legally binding contract, and that can enforce their terms and conditions within that contract, regardless of the blockchain platform. Taking into consideration the requirements that are formulated in Chapter 1, Section 1.5.2 this study sets out to investigate a method that allows for the design of smart legal contracts. From the discussion in this chapter and thus the overall research we can draw several conclusions.

Firstly, we posit that MDA is a suitable philosophy to develop smart legal contracts. The MDA philosophy allows for the generation of platform agnostic smart contract code by employing transformations between models thus satisfying requirement 5 while also addressing requirement 6. The transformation from the concepts in the domain ontology present in chapter 5.7 is more problematic, and results in our second observation. Secondly, we observe that problems related to using smart legal contracts to either complement or replace legal contracts predominantly emerge during two life cycle phases of a smart legal contract. The first set of problems arise when translating concepts of a legal contract to smart contract code. During the construction of the domain ontology we found that generic concepts in legal contracts like actions, clauses and assets can cover most part of the translation from a legal contract. However, we also found that several types of modalities, actions, assets and clauses should be discerned. Each class of these concepts requires a different code pattern for each of their respective types. The second set of problems emerge during the monitoring and enforcement phase of the smart legal contract. A noteworthy observation from the discussion in this chapter is that some problems like what actions the smart contract should monitor have relation to the first set of problems occurring during translation. The second observation that we make with regard to letting smart legal contracts enforce legal contracts is that especially once disputes arise, or when the contract is breached currently there are few code patterns that cater for such events.

Thirdly, the results of our study demonstrate that the development of platform agnostic smart contracts is feasible yet hard to achieve. For our research we reconciled concepts that are used to write Ethereum and Hyperledger smart contracts into one platform agnostic metamodel. What strongly sets Ethereum and Hyperledger Fabric smart contracts apart is the manner that both platforms have implemented the notion of access control and data

privacy. Although it is possible when writing Ethereum smart contracts to specify whom can interact with a set of functions, there is no direct method to guarantee the privacy of the code or of the transactions resulting from the execution of the smart contract. In contrast, for smart contracts these options are available to users. The literature review in Chapter 2 shows however, that ZK-SNARKS are further being developed to remedy these problems for Ethereum smart contracts. An important consideration that this research presents is whether it is worthwhile to develop a platform specific metamodel for all smart contract supporting blockchain platforms currently in existence. We experienced that creating a smart contract metamodel requires vast amounts of expertise and time.

Fourth, achieving full traceability as stipulated in Requirement 4 could be fully addressed by designing routines that resolve potential translation and runtime problems. However, working out and constructing routines or mechanisms requires additional multidisciplinary research efforts that demand the involvement of several stakeholder groups and thus is for now beyond the scope of this research. We would however, like to emphasize that some invaluable efforts have already been undertaken in this direction. In section 7.4 we suggest that scholars and practitioners should focus future research on devising such mechanisms or routines.

Chapter 8

Conclusion

8.1 Introduction

The final chapter of this dissertation draws conclusions on the findings and discussion we conducted. This chapter is hereafter organized as follows: Firstly, the chapter will recap the research conducted for this dissertation in light of the purpose we have defined and highlight main observations. Thereafter in the second section (Section 8.3) we state the theoretical contributions of this research. Continuing, Section 8.4 expounds the practical implications of this research.

8.2 Overview of the Research and Main Observations

The primary purpose of this research is to develop an understanding of what *method* would facilitate the development of smart legal contracts that are legally binding and can (partially) self-enforce conditions within a legal contract, regardless of the blockchain platform. In this dissertation a method is presented for this purpose founded on the MDA philosophy. In chapter 1 we argue why this study is warranted, and Section 1.5.2 further discusses the requirements for the method. Given that the problems identified for this research stem from legal and technical stakeholders, a solution requires to be satisfactory from both the legal and technical the perspective. There are a myriad of blockchain platforms that facilitate the deployment of smart contracts. A review of the state-of-the-art reveals that most of not all methods currently employed to develop smart legal contracts are focused on one blockchain platform. Therefore, another requirement deemed appropriate for the method is that it enables the development of blockchain platform agnostic smart contracts. On the other hand, the method needs to facilitate the creation of a smart legal contract on a specific platform. Legal professionals most likely will not be able to understand code, and programmers might not have the required legal expertise to directly translate the legal requirements into code. Therefore, another important requirement for the method is that domain knowledge can be made explicit so both groups of stakeholders can understand each other. The use of smart contract within the legal domain introduces several challenges. Each of these challenges is related to distinct parts of a smart legal contracts' life cycle.

In the second chapter the extant literature on BCT is discussed to create an understanding of the technology underpinning smart contracts. The chapter presents a FCA [175] that defines BCT, provides a systematic overview of the of the art concepts in the domain and distills a grounded research roadmap. Using the well-known 4+1 software architecture viewpoint framework [141], the architecture elements of BCT were fleshed out, specifically, our results recap: (1) the way a platform can be designed: (2) how transactions are processed and, (3) the architectural arrangements typically used for the P2P network underlying BCT. The

chapter also expounds on the architecture properties of blockchain technology and their trade-offs. Data analysis reveals 8 coupled architectural characteristics — these properties are trade-offs exercised during blockchain architectural design. The fifth objective of the research conducted for the chapter was to identify the challenges for BCT. In the future, 6 main challenges for blockchain technology need to be addressed: (1) decreasing latency for the conformation of transactions; (2) increasing the throughput of transactions which is related to the design of the consensus protocol; (3) decreasing data storage requirement; (4) protecting the privacy of blockchain users; (5) data governance of blockchain networks; (6) the usability of the technology. Finally, an analysis of the papers under review demonstrates that there are four research gaps that need to be addressed by future research concerning: (1) consensus protocols, (2) Data privacy and storage, (3) smart contracts, and (4) the usability of blockchain for end users.

Further background on smart contracts is provided in chapter 3. The chapter exemplifies the use of smart contracts using a motivational example. From our investigation into smart contract supporting blockchain architecture we can conclude that (1) smart contracts are strongly tied to the infrastructure they are deployed on. The infrastructure usually consists of six layers including: the infrastructure layer (blockchain), smart contract layer, business service layer, business transaction layer, business process layer and finally a business process layer. (2) blockchains are self-contained systems that allow for the inclusion of some external data from Oracles. However, oracles introduce several problems when used. (3) most smart contracts are currently employed for business transactions with a life-cycle different from smart legal contracts. This difference can be attributed to the fact that legal requirements imply the inclusion of additional phases like a dispute resolution and termination phase. In chapter 3 a case study further delineates the use of smart contracts for a simple transaction scenario. Something that can be observed from this case study is that most actions within the smart contracts are executed by human agents and not the smart contract.

Chapter 4.3 discusses the MDA philosophy that underpins the method presented in this dissertation. For the employment of MDA to develop software two metamodels are at least needed: (1) a platform agnostic model (2), and a platform specific model. The former allows for the abstraction of common concepts shared across platforms, regardless of the platform. A specification of a platform independent metamodel allows users to create PIM's that capture the design of their application from a platform independent perspective. The latter encompasses concepts that are used to write code for a specific platform. Transformation rules enable the transformation from the abstract platform independent model to a platform specific model. Another optional third metamodel that can be used in concert with a PIM and PSM is one from a computation independent perspective. Several types of metamodels are employed for this purpose like domain ideologies and domain models. Ultimately all of these models have the same aim of providing end-users a manner to stipulate a CIM that depicts concepts relevant to their domain. Although employing MDA to develop software has its merits, we found that it also has some disadvantages. (1) models are simplified representations of a certain reality. This simplification inherently results in a less detailed depiction of that reality when developing software. (2) the modeling language used to model a metamodel determines the expressive power of that model. (3) Once the metamodels are constructed developers can reap the benefits of the method. Initially however, a significant amount of expertise is required to create these models. (4) Although the number of tools available for MDA has increased over time some caveats remain. For instance, tools to transform a CIM into a PIM are not supported. Despite these disadvantages our research suggests that employing the MDA philosophy to design smart contracts increases productivity and promotes interoperability of models. Combined, these we thus formulate the propositions

that MDA increases productivity and promotes interoperability of models when developing smart contracts.

In chapter 5 a domain ontology for smart legal contracts is presented that can be used to specify the design of a smart contract. By reviewing the extant literature on legal requirements engineering, MAS and e-contracts we have identified several concepts required to model smart legal contracts. Examples of these concepts are the notion of a contract, between party's that own assets. Clauses govern the behavior of the party's and contain the (conditional) promises that the parties agreed upon to perform. Each promise must be performed by a party by undertaking an action. The domain ontology is not without limitations. Several types of assets, clauses, action groups can be discerned. However, the current study does not provide an extant or exhaustive list of types for each of these concepts. Besides these limitations the ontology currently lacks the possibility to specify that a certain agent is authorized to perform an action. Scholars and practitioners have suggested that employing smart contracts to represent legal contracts introduces several challenges. Being aware of these challenges, this dissertation presents a total of 21 challenges that are identified through a literature review. In turn, these challenges are coupled directly to concepts of the domain ontology. The discussion in Section 7.2.3 suggests that these challenges are not easily to overcome as it is difficult to codify them, meaning to structure them into code. This leads us formulate the research proposition that legal contracts with a lower codifiability, are less suitable to translate into a smart legal contract. This same discussion also prompted us to formulate the proposition that when less contingencies are specified in a legal contract, the less suitable it is to translate it into a smart legal contract.

Chapter 6 presents a platform specific model for the two most prominent blockchain platforms: Ethereum and Hyperledger Fabric. Following a collaborative method, we developed the platform specific metamodels for these platforms with the help of 11 blockchain experts. During several consecutive rounds in a Delphi study, we identified all relevant concepts requires to build the platform specific metamodels. Thereafter the research has set out to create a PIM based on the commonalities between the concepts that both platforms use. The comparison identified a large set of commonalities. However, some distinctions were also identified. Among these distinctions is the difference how user interaction with the smart contracts is restricted by preconditions. Whereas Ethereum is a public blockchain meaning that transactions and smart contracts can be publicly audited and actions be performed by anyone, Hyperledger Fabric smart contract developers can restrict the operations of a user depending on their role. Examples of these operations are creating new assets, update the state of an asset, or restricting the visibility of a transaction. Another dissimilarity between the platforms is the manner smart contract can obtain data concerning their own state(s) or that of the blockchain. The chapter demonstrates that the design principles of private and public blockchain platforms that support smart contracts can be reconciled into a PIM.

8.3 Theoretical Contributions

The the findings of this research suggest that MDA is a method suitable for the development of smart legal contracts. When developing software based on the MDA philosophy meta-models are required to enable the translation process. The creation of these models has yielded interesting theoretical insights that can be employed to advance our knowledge on BCT and smart contracts. From a theoretical perspective, this research contributes to the theoretical bases of blockchain and smart contract literature in several ways:

First, in Chapter 2 we systematically review the extant literature to create an understanding of BCT architecture. This understanding aids in theorizing blockchain architecture and identifying commonalities between different blockchains. Furthermore, we discuss the challenges and characteristics of the technology. Identifying the challenges raises awareness on the potential challenges that BCT faces. By determining challenges and characteristics in concert some important trade-offs emerge. Information with regard to these trade-offs can be employed to inform future research. In addition, the chapter provides a research roadmap that assists practitioners and scholars in addressing current research gaps.

Second, understanding the relation between BCT and smart contracts is pivotal as the former providing the infrastructure underpinning the latter. In Chapter 3 of the dissertation this relation is further clarified and exemplified. The relation is clarified by presenting and discussing the layered architecture of BCT and how it supports smart contracts to enhance the present knowledge on the topic. The discussion of the architecture offers researchers an integrated perspective on smart contract supporting blockchain architecture. By further explaining the machinations of smart contracts using a case study we provide a practical insight into the use of the technology.

Third, this work presents a domain ontology that can be used for several types of legal contracts. Therefore the domain ontology provides the foundation for a unified perspective on smart legal contracts. We argue that a unified perspective allows for more abstract reasoning on the use of smart legal contracts in legal practice. Furthermore, by providing this unified view a common understanding of the domain is cultivated within otherwise desperate fields.

Fourth, the disciplines of computer science and law are complex and both have their own nomenclature [57]. The ontology enables programmers and legal professionals to communicate with one another by providing a lingua franca for the domain. Therefore the ontology eases and enables the communication of advances in research among the disciplines. Crucially the ontology serves as a stepping stone for further inter-disciplinary research that in the past has hampered similar efforts.

Fifth, even though prior work has addressed the issues of understandability or made efforts to provide a faithful representation of legal concepts, legal realism is still lacking. Legal scholars have espoused various problems when translating legislature to code [125], or more specifically legal contracts to smart contracts [74]. In the works most related to ours these problems are not taken into consideration. Indeed, most practitioners and scholars that advocate the idea of smart contracts assume that it is always possible to draft complete and unambiguous contracts [179]. Contrary to prior literature we acknowledge the existence these problems and relate them to the concepts of the ontology presented in this study. By laying bare the legal challenges related to smart legal contracts, the ontology aids practitioners in predicting, addressing, and specifying potential issues. Besides, researchers will be better able to direct their research efforts towards solving these problems as they are explicit.

Sixth, the platform independent metamodel in chapter 6 provides an overview of smart contracts that goes beyond piecemeal initiatives to investigate the inner workings of smart contract for a specific platform. By identifying the shared commonalities of smart contract supporting blockchain platforms this work contributes to an understanding of a generic perspective on the matter. Generalization enables further theorizing of smart contracts and thus easing the identification of problems across platforms.

Seventh, in spite of its limitations, the study adds to our understanding of the limitations and problems related to employing smart legal contracts for legal contracts. The combination of models in this work delineates a potential method to develop smart legal contracts that sheds a light how this could be achieved. The emerging limitations and gaps that this

research has are employed to suggest a framework for future research that we hope invokes a call to action.

8.4 Practical Contributions and Implications

From the results of this study, we also identify some practical implications. An important practical implication of using blockchain and smart contracts is that parties no longer rely on trusted third parties to execute their transactions, but algorithms and code. Consequently, if the parties do not trust blockchain technology or smart contracts they will most likely refrain from using it. Widespread adoption of smart contracts will only take place once parties and other stakeholders that operate on behalf of these parties trust the technology. However, as Al Khalil et al. [3] suggest, trust in smart contracts is a process as well. This research contributes to the growth of this trust in several ways.

The disciplines of computer science and law are complex and both have their own nomenclature [57]. Our work presents a domain ontology that can be used for several types of contracts. One the practical contributions of this work is that the ontology enables programmers and legal professionals to communicate with one another by providing a common understanding of the domain. We hope that this common understanding will enhance the mutual trust between legal experts and programmers. However, one implication for practice that this research lays bare is that to further the development of smart legal contracts in the future far more cooperation between legal experts and programmers is needed to further foster trust.

Legal experts need to trust the ontology to be sound and relevant for their work. Therefore, it is important to engage legal experts in future development efforts. Besides that, engaging legal experts in the development process further enrich the ontology with relevant concepts, it might also lead to a stronger support base and thus broader adoption of the ontology. A broad adoption of the ontology is needed if smart legal contracts are to be accepted as a (partial) legitimate substitute for legal contracts used nowadays.

A domain ontology with a strong support base among legal experts, offers a solid foundation for the realization of standards. Standards are crucial to foster trust among users. Most blockchain platforms openly discuss proposals for standardization and anyone is welcome to submit a proposal¹. We also note that disparate fields have addressed different aspects of e-contracting. This has led to a fractured perspective on electronic contracting and the requirements thereof. In the same vein, hitherto three noticeable but disparate efforts have been made to standardize smart legal contracts. Among these are the Common Accord project, Accord project and the work of Ladleif and Weske [150]. A unified research effort directed towards the development of a standard for smart legal contracts that can be implemented across platforms is needed. This is important as without a unified standard organizations may be inclined to postpone the adoption of smart legal contracts until a dominant design has established itself [74]. The models presented in this thesis present the first step toward this standardization.

Even though prior work has addressed the issues of understandability or made efforts to provide a faithful representation of the concepts used in legal contract, legal realism is still lacking. Legal scholars have espoused various problems when translating legislature to code, or more specifically legal contracts to smart contracts. Contrary to prior literature we acknowledge the existence of these problems and relate them to the concepts of the ontology

¹See for instance the Ethereum EIP site where proposals are discussed, and the site where Bitcoin proposals are discussed.

presented in this study. By laying bare the legal challenges related to smart legal contracts, the ontology aids practitioners in predicting, addressing, and specifying potential issues. Ultimately, this makes for more realistic and trustworthy smart legal contracts.

Other practical contributions of this study are the platform agnostic metamodel for smart contracts and the platform specific metamodels for the Ethereum and Hyperledger Fabric platforms. The insights that are obtained by creating these models show that most concepts for both platforms can be reconciled into one platform agnostic metamodel. From a practical perspective this means that although programmers need to learn distinct programming languages to write smart contracts for these platforms the concepts employed are similar. The differences between the platforms that we identified also provide useful insights for practice. When coding smart contracts for the Ethereum platform developers will have to define their own access control rules to restrict and enable access to a smart contracts' operations. These aspects are more easily defined for Hyperledger Fabric smart contracts yet as our research shows the blockchain on which the smart contract is deployed is more complex.

Appendix A

Research Methodology Blockchain Technology

This appendix provides a full description of the methodology used to attain the results presented in Chapter 2.

A.1 Formal Concept Analysis Methodology

For this research we aim to define BCT from a software architecture perspective based on the extant literature (as stated in research question 1). Hence, the objects used for conducting the FCA were all GL and SL items selected for this study. We first selected the definition of software architecture provided by Bass, Clements, and Kazman [25]:

"The software architecture of a system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."

We have perceived software elements as the main attributes of BCT. The FCA has been conducted in the following steps:

1. Each selected item has been reviewed to identify all software elements of BCT (e.g. nodes and block) stated in the introduction or background sections using the codes that were generated during the open coding process of the GT approach.
2. Whenever a software element was identified it was given a specific code (BCTEL).
3. All of the identified software elements were identified for the FCA were placed in a matrix. The rows of the matrix represented the GL and SL items whereas the columns represented the software elements that were identified in these items. If one of the items described a particular software element an X was placed in the corresponding cell.
4. From the analysis a pattern of 10 attributes emerged that were most recurring. These attributes have been used to construct a definition of BCT.

A.2 Grounded Theory Analysis Method

Our GT coding process encompassed the following phases:

1. Open coding - (four phases). During the first phase of the open coding we have conducted a pilot study: Fifteen primary studies from the selected items have been randomly selected to establish an initial set of codes using open coding. A second pass has been made on the pilot papers to generate a final list of codes, in order to minimize inconsistencies during the coding process. In the second phase, using the priorly established set of codes, an initial theory about the relation between codes has been developed based on the pilot study. The third phase consists of constant comparison:

the pilot study initially generated 266 codes. Next, these initial codes were organized into a hierarchy of codes based on emerging relations between concepts. The resulting structured start-up list of codes was used to code the remainder of the primary papers. The coding has been executed in parallel by two different coders, over two equally divided splits, to ensure avoidance of observer bias. Each primary paper has been analyzed line by line with the list of codes. Codes were applied if they reflected a concept in a paragraph. For example, the paragraph: *"An important component of the blockchain technology is the use of cryptographic hash functions for many operations, such as hashing the content of a block. Hashing is a method of calculating a relatively unique fixed-size output (called a message digest, or just digest) for an input of nearly any size (e.g., a file, some text, or an image)."* would be coded with the code "FUNC-HASH", where FUNC denotes the code's reference to functional aspects of blockchains while the -HASH refers to a refinement of such functional aspects to refer to hash-functions, in the specific. The fourth phase, constant memoing, has been conducted simultaneously with the third phase. During this phase we have kept notes to capture key messages, relations and observations on the analysed texts.

2. Axial coding - (two phases). In the first phase comparing the concepts coded has led us to inductively generate relations among coded concepts (e.g. between consensus protocols and blockchain challenges); For the second phase, the definitions of all concepts coded have been compared with each other to identify aliases.
3. Selective coding - (three phases). The third step involved the selection and arrangement of codes to form a relationship model. As a first phase have we arranged the data: Every portion of text that was coded with a certain code has been placed in a table. Each of these codes represented a core concept observed in the literature (e.g. consensus protocol, blockchain networks and applications). Next, in the second phase we have modeled the data. The data was represented in a view consisting of several diagrams. Whenever possible these diagrams were connected to one another, resulting in the construction of the views of BCT architecture. Finally, as part of this phase, the diagrams and all the data at hand has been analyzed and sorted to address the research questions behind this study.

A.3 Inter-rater process

The first two authors of this paper examined their inter-rater reliability via the following approach: First, the inter-rater reliability of the assessment of the SL items was determined. The Krippendorff test revealed that there was 86% inter-rater agreement (α 86%). Second, another Krippendorff α test was employed to determine the inter-rater reliability between both observers with regard to the in- and exclusion criteria for GL. The resulting K- α statistic (α 93%) showed that there was a high inter-rater agreement on the exclusion and inclusion of GL items. Third, an inter-rater reliability of the analysis that was conducted concerning the quality of the GL studies was tested using a Krippendorff α test. The results of the test indicated that there was a 99% agreement (α 99%) between both observers. Finally, the findings presented in the sections hereafter were attained by adopting a thematic coding process. Therefore as a precautionary measure to avoid observer bias during the coding process we conducted an inter rater reliability assessment test to the codes that were obtained from the pilot study. The results of the Krippendorff's α test suggest that there was

a 69% agreement between the observers. Because this result was below the commonly accepted threshold of an α of 80%, the first two authors deliberated over the differences to form one consistent initial coding set.

Appendix B

A 4+1 Architectural View of Blockchain Technology

This appendix contains additional material that has been written for the research concerning the architectural design of blockchains. Taking on different perspectives the viewpoints of a blockchain architecture are discussed in this Appendix.

B.1 A 4+1 View of Blockchain Technology

B.1.1 Logical View

The logical view emphasizes on the functional requirements and services the system should provide to its end users [141]. Decomposition of the architecture aids in identifying the elements that are common across the system. We used an ontology for BCT as proposed in [94, 143] to organize the discussion of its main elements.

B.1.1.1 Transactions

Transactions in a blockchain system are executed using *public key cryptography*. Actors that send transactions between themselves and other beneficiary actors. Two pairs of keys are used to allow actors to interact with one another; a private and a public key that are mathematically related to each other [14, 17, 53, 278].

Actors can use their secret *private key* to sign transactions, that are addressable on the network via their public key. Private keys are stored in software called a *wallet* that is installed on a hardware device. A wallet can also store public keys and associated addresses to send transactions to [189, 278, 280]. The *public key* is widely disseminated without reducing the security of the transaction process [115, 278]. Public keys have various functions they are used to derive addresses and to verify the signatures generated with the private keys [181, 278] (e.g., the Elliptic Curve Digital Signature Algorithm (ECDSA) [183, 254, 280]). A transaction transfers an amount of coins from one *input address* owned by the private key owner to one or several *output addresses* [219]. An *address* is created using the public key, the private key, and a cryptographic hash function [68].

Cryptographic Hash functions are used for the purpose of many operations, such as signing transactions (SHA-256 in the Bitcoin case[188]). *Hashing* is the process of converting any data of arbitrary size to data of fixed size where the output is a bit-string known as the digest, hash value, hash code or hash sum [173, 189, 209, 278]. Tampering with the original transaction data would immediately get noticed as the hash would differ from that previously generated and recorded on the blockchain [14, 173, 189, 194]. Hash algorithms are generally designed to be *one-way*, meaning that one cannot compute and extract original data from a hash [14, 173, 193, 278].

B.1.1.2 Nodes

The peers in the P2P network, also referred to as nodes are devices capable of processing and verifying transactions. Depending on the permissions all nodes or a specific subset of nodes validate transactions. The *permissions*, refer to the rights that are granted to the nodes in a blockchain network. There are three major types of permissions [112]: *Read permission* that dictates which nodes can access the ledger and audit transactions, *Write permission* that stipulates who can create transactions and broadcast them to the network, and *Commit permission* that describe who can update the ledger. Some nodes may only have permission to use the services in a blockchain network such as announcing an transaction, while other nodes are permitted to propose to include certain unspent transactions in a block to update the ledger [212, 259, 272]. Other blockchains (e.g. MultiChain [97]) allow for more fine grained permissions, for example the *permission to create assets* [112, 231, 274, 278].

Updates and changes to the software of a blockchain are called *forks* [189, 278]. During the mining process two miners might propose a block at the simultaneously. This type of fork can be described as an *accidental fork* that stems from the probabilistic nature of the mining process [189]. To resolve the fork the miners as a rule follow the longest chain. The longest chain rule can also effectively be employed to invoke two other types of forks and propose updates to the blockchain [189]. The first of these types of forks is a *hard fork*.

A hard fork brings about updates to the BCT that prevents nodes that do not accept the fork from using the changed BCT or the nodes can continue to use the original protocol without the update. Nodes that use dissimilar hard forks cannot interact. Structural changes to a blockchain requires hard fork [278] and can also effectively create a new blockchain [274].

The second type of fork is a *soft fork*. Only a super majority of nodes need to upgrade to implement the novel rules stipulated by the soft fork. Nodes that did not update still accept newly created blocks as valid after the soft fork [18, 278]. At a later point in time the two chains are reconciled with one another. The possibility to make updates to a blockchain or distributed ledger implies that there are also *Update permissions*.

There exists at least three categories of blockchain networks [274, 285]; Public, private, and consortium networks that have different arrangements in terms of their permissions. A node that has permissions and function to verify transactions that are executed on a private or consortium blockchains is a *validator* [44, 62, 143, 193, 247]. On a public network a node can also validate transactions by proposing blocks [72]. Nodes that fulfill the aforementioned are also called *miners* or *forgers*. Miners or forgers have two separate functions; (a) to correctly construct and propose new blocks, and (b) to check the validity of the transactions in each block [163]. In the specific case of some blockchain platforms (e.g. Ethereum) that allows for the creation of smart contracts, miners, forgers and validators have the additional task of executing the smart contracts to check for the validity of their outcomes.

B.1.1.3 Blocks

On a blockchain transactions are stored in *blocks*. Blocks can be divided into two parts [183, 280, 282]; (1) a *blockbody* that contains the verified transaction data, which is recorded in the form of a Merkle tree [274, 282], a Patricia Merkle tree [72, 93] or Bucket hash tree [72], and (2) a *blockheader* that specifies the elements required to guarantee safety. Rather than storing the hash of each transaction individually these are stored in a data structure known as a Merkle tree to further diminish the storage requirements.

A Merkle tree combines the hash values of the transaction data by re-hashing them until there is a single root left which is called the *root hash* [72, 188, 278, 285]. Since the root

hash is a hash function it can be used as a mechanism to summarize transactions that have been stored in a block, any alterations to the underlying transactions would be detected [94, 274, 278]. Some blockchains employ other means of data structuring that also facilitate in capturing smart contract states besides transactions. For instance, Ethereum employs a Patricia merkle tree whose leaves record key-value states [72]. Hyperledger implements a Bucket-Merkle tree that groups states into a pre-defined number of buckets [71]. The maximum number of transactions that a block can contain depends on the block size and the transaction size [183]. The blockheader encompasses six elements [115, 183, 282, 285]; (1) a root hash, (2) a block version number that specifies the software version of the block, (3) Blockheader hash of the previous block, (4) the timestamp of the block, (5) the difficulty (target) required to create the block and; (6) a nonce random number.

B.1.1.4 Chains

Each block is linked to its predecessor known as *parent block* by including its blockheader hash to form an integral chain of blocks that can be traced back to the first, or *genesis block*. Hence the term "blockchain" technology. By hashing transactions and chaining all blocks to one another a blockchain provides a data model that allows to track all historical changes to the distributed ledger. Moreover, by combining the hashing of transactions and the chaining of blocks the distributed ledger becomes tamper proof¹.

Novel blocks are generated using a consensus protocol. Provided that the transactions included in the newly proposed block are valid, each new block enhances the security guarantees of the block before it [183, 188, 189]. The number of blocks in the chain between the last created block and the genesis block is called the *block height* (So a genesis block has height 0.). Because of the probabilistic nature of some consensus protocols (e.g. PoW and PoS) two miners can propose a valid block simultaneously; This can result in the situation where *stale blocks* are created which will never be included in the longest chain, and can therefore be considered as wasted efforts [163]. The Ethereum platform refers to these blocks as *uncle blocks* [61, 285]. Forks that occur accidentally or are invoked with ill intent are resolved using a *fork choice rule function* [212] (e.g. longest chain rule). Eventually, one of these chains will become the longest chain and the other shorter chains will be abandoned. Blocks that are included in the abandoned are colloquially known as *orphan blocks* [212, 259].

B.1.2 Development View

The deployment viewpoint defines how the various elements identified in the logical, process, and implementation viewpoints mapped onto the various nodes [141].

Every node in a blockchain network has two layers; an application layer and a blockchain layer [277]. Therefore, there are two groups of developers to be considered: First, Developers utilizing the services of a blockchain platforms (e.g. Bitcoin or Ethereum) to develop decentralized applications. Second, developers that seek to create a new blockchain platform.

B.1.2.1 DApp developers

The blockchain networks can be leveraged to build *decentralized applications* (DApps) that use their services. DApps are applications designed have a distributed nature as they are run on a P2P network instead of one computer. At first sight a DApp looks similar to that

¹According to Stark [243] this term needs to be qualified because there is a very high probabilistic guarantee that the data recorded on the blockchain is not changed.

of a normal (web) application. Instead of using an API to connect to a database, a smart contract will connect the DApp to the blockchain for all required information. The front end of a DApp can therefore be regarded as a facade that allow users to interact with the services provided by the blockchain platform. Using the smart contract services of blockchain networks such as Ethereum allows developers to create custom DApps suitable for a wider range of applications. Developing a DApp might include coding a smart contract that needs to be compiled using a virtual machine that is offered by a blockchain platform. The final step is to deploy the smart contract on the blockchain network by using a wallet. Once the final step has taken place the DApp can interact with the blockchain platform by invoking smart contracts.

B.1.2.2 Establishing the P2P network

Developers seeking to build their own blockchain platform have to program multiple software packages. Enabling transactions forms the basis for any blockchain network. A wallet needs to be programmed to allow clients of the platform to interact with other peers in the network. In order for peers on the networks to interact they need to be knowledgeable about one another. Therefore there should be a *address propagation* method installed. Next the nodes need to connect via *peer discovery*. Peers of the Bitcoin network for instance, connect to each other over an unencrypted TCP channel. Every node keeps a list of IP addresses associated with its connections since there is no authentication of nodes. However, for networks that consist of peers that already know each other a different approach to peer discovery might be possible. Another aspect is the mechanism for *propagating transactions* [33]. The Bitcoin uses a propagation mechanism where the nodes in the P2P network forward transactions to their neighbors (known nodes). Other platforms such as Ripple [226] use a pre-defined node list that a node must store to process transactions.

B.1.2.3 Data Model

Data with regard to transactions can be stored in two ways: As a first method, like the Bitcoin, one can choose to add data into transactions. Another second method is to add data into contract storage like Ethereum [277]. The state of the distributed ledger is a collection of all the accounts that have not been spend yet, referred to as the *unspent transaction output model* (UTxO) [115, 285]. The Bitcoin can therefore be regarded as having a transaction based model. Another choice that could be made is to offer smart contracts as a service. In these instances the transactions resulting from the execution of smart contracts need to be stored. Ethereum stores smart contracts in specific accounts using an *account based data model* [72]. The state of transactions in the system are the changes of the complete contract storage expressed as key-values [277]. Other platforms such as Hyperledger simply use key-values to store data in Docker containers [72].

B.1.2.4 Consensus protocol

A consensus protocol can be regarded as a sequential set of steps that stipulate the rules of engagement for the network to process transactions and smart contracts. Various consensus protocols are available for this purpose with idiosyncratic properties in terms of throughput, latency and security. However, developers can also opt to design a novel consensus protocol from scratch. To verify transactions during a consensus protocol the wallet should be able to import a copy of the distributed ledger. To compile the smart contracts written by users of the platform into bytecode that can executed by the platform, the creation of a virtual machine might be needed. This virtual machine is also used to verify the outcomes of the

execution of smart contracts. Platforms like Kadena use an interpreter language to write smart contracts which makes a virtual machine redundant. Permissions should be allocated and distributed among nodes to confine permissions of nodes to participate in the consensus protocol if desired. A mechanism to distribute economic incentives need to be devised for nodes that verify transactions [183, 277, 282].

B.1.3 Process view

The process view specifies which thread of control execute the operations of the classes identified in the logical view. It also takes into account non-functional requirements such as performance and system availability. The consensus protocol is at the heart of all BCT processes since it allows for the enactment of transactions and ensures that the distributed ledger remains consistent. Consensus protocols are discussed in chronological order.

B.1.3.1 Practical Byzantine Fault Tolerance

(PBFT). PBFT is mostly used in a private setting for permissioned blockchains because it assumes authenticated nodes [72, 274, 285]. The protocol itself is exclusively based on communication, and nodes go engage in multiple rounds of communication to reach consensus [72]. Nodes do not get a reward for achieving consensus, rather in the event of malicious behavior by an authenticated node it can be held legally accountable [112, 209]. A primary leader node mines the blocks. The leader can be changed by other nodes via a "view-change" voting protocol, in the occurrence of a crash or when it exhibits malicious behavior [53, 183]. PBFT as has five phases to reach consensus [183]:

1. *Request*. A client sends a request for a transaction to a leader node, that accordingly gives the request a timestamp.
2. *Pre-prepare*. The leader node records the request message and renders an order number for it. Then the leader node broadcasts a pre-prepare message with a value to the other nodes. Initially the nodes decide on whether to accept the request or to reject it.
3. *Prepare*. When a node decides to accept the request it broadcasts its values in the form of a message to other nodes. Nodes then receive prepare messages from one another. Once a node has collected sufficiently enough messages from other nodes ($2f+1$ messages), hence if a majority of nodes decides to accept the request, it will enter the commit phase.
4. *Commit*. All nodes that are involved in the commit state send a commit message to the other nodes. Concurrently, if a node receives a message of acceptance from $2/3$ of the other nodes in the network a consensus has been reached to accept the request. Next, the node executes the instructions that are stated in the request message.
5. *Reply*. The nodes in the network reply to a request of a client. If a delay in the network occurs and the client did not receive a reply message, the request will be resend. In the case that the request did get executed the nodes in the network need to send a reply message repeatedly.

The protocol require nodes to have a high degree of trust and is not suitable for permissionless blockchain. PBFT is purely communication based rather than on cryptographic primitives. Therefore, whenever an authenticated peer is attacked, the blockchain could easily be manipulated [173]. PBFT does not scale well as executing the PBFT consensus protocol can

be time consuming. As more nodes join the network the latency increases and throughput decreases [183].

B.1.3.2 Proof-of-Work

(PoW) is often referred to as the *Nakamoto consensus protocol* [121, 163, 164, 204]. Recent estimations indicate that the majority of public blockchains use PoW as their mechanism to create a consensus [71, 173]. Public blockchains need to have a high degree of Byzantine fault tolerance as users can not trust one another. The PoW consensus protocol is designed for the case where there is little to no trust amongst users of the system [278]. Furthermore, it safeguards against Sybil attacks which are common in open, decentralized environments in which a malicious actor can acquire multiple identities [71].

Consensus in PoW is achieved through a hashing competition between miners. Competing miners need to commit computing power to calculate the solution to the same mathematical problem. To incentivize miners to participate in the consensus process the miner that is the first to find the solution to the mathematical problem reserves the right to publish the next block, and is rewarded by an amount of cryptocurrency² [36, 183, 274, 278]. In addition, the miner to win the competition with its peers is also able to collect the transactions fees that were paid by clients³.

Finding the solution to a PoW problem is a computationally arduous process for which there are no shortcuts [189, 278]. The solution to the problem is hard to find, yet easy to check once they have been found [163]. Given that only one miner can win the competition and is rewarded the other nodes have simply wasted resources (CPU power and energy) in their attempt [88, 183, 244, 274, 278]. In addition, because the difficulty of PoW problems increases over time makes it even harder to win the competition [278]. Some miners have as a result started collaborations called *mining pools* to address this problem. When participating in mining pools miners combine their computing power and divide the work to find the solution [247]. The rewards for proposing the next block are split based on each node's contribution [121]. However, pooled mining could open the opportunity for a 51% attack if a cartel of large mining pools were to control more than 50% of the hashrate [183, 247]. The large accumulation and *centralization of hashing power* could constitute to a majority vote of miners which could endanger the integrity of the ledger. The steps in the mining process are as follows [84, 85]:

1. *Get the difficulty.* The difficulty of the PoW problem changes every *epoch* (fixed number of blocks) and depends on the *target* that is generated for each block [85, 183, 278, 287]. Therefore, a miner first needs to use a *target recalculation algorithm* to determine the correct target for the problem, which depends on the hash rate of the whole network. The *hash rate* is the number attempts required to solve the PoW [178, 183].
2. *Collecting Unspent Transactions.* All requested transactions that have not been processed since the generation of the last block are stalled in a *transaction pool* [212, 278]. During mining, miners collect and verify all of the transactions that are kept in the transaction pool and hash them to a root hash. To mine the block the miner also needs to include the block version number, blockheader hash of the previous block, nonce random number and the target found during step 1 [183].

²In the case of Bitcoin the amount of cryptocurrency that is provided to miners for their services decreases to zero over time as it halved every 210,000 blocks [53]. From that point on, miners can only be compensated by transaction fees alone [188]

³Note that transaction fees are not obligated to execute Bitcoin transactions but rather have been paid by clients to prioritize their transactions [247].

3. *Calculating the blockheader hash.* Next miners start their attempts to solve the mathematical problem which comprises finding the correct nonce so that the *double hash value of the blockheader* (SHA-256^2) is equal to or smaller than the number of zeroes (target) that the network demands [53, 183].
4. *Broadcasting the Block.* If a miner has found the solution it is propagated along with a *candidate block* to the remaining nodes in the network.
5. *Verification of the Block.* Upon receiving a candidate block other nodes will check (a) whether the candidate block includes correct hash references to the previous block on their chain and thus, the proposed nonce to solve the problem [189]. Further, (b) whether the block contains valid transactions by verifying the root hash [278]. If the block is found to be valid nodes express their acceptance by appending the block to their copy of the blockchain, and resend the block to other nodes. Finally, when a consensus is reached by a majority of nodes in the network (51%) that the block is valid it will be included in the chain of blocks.

Whenever a fork occurs miners will wait for longest chain to be formed by the addition of new blocks at the cost of high latency. The system is therefore designed under the presumption of *eventual* reconciliation of the ledgers, which can be regarded as non deterministic [72]. PoW ensures a high degree security as theoretically it has a byzantine fault tolerance of 51% [72, 89, 183]. Some scenarios to attack for PoW attacks have been identified such as a *selfish mining strategy*. In this scenario when miners mine a correct block they do not propagate it. Rather, they privately mine blocks and hide them. The private branch of the chain will only be revealed to other nodes under the right circumstances during which it can result in a higher of frequency of forks that pose a threat to the blockchains security [84, 89, 183]. Another attack scenario is the verifiers dilemma [163] wherefore miners do not receive any reward for validating the correctness of a block proposed by another miner. This is because it is assumed that the other miners want to maintain a correct blockchain so that the value of the Bitcoin does not depreciate. However, this assumption creates a dilemma for miners whether or not to actually check the validity of the transactions included in the block as they do not receive any reward for doing so.

Most PoW consensus protocol variants have a low throughput [240]. This problem is further amplified because PoW based blockchains also has a relatively high latency (time it takes to confirm the transactions) due to the fact that the block time interval is set at a fixed. This is because between the propagation of two blocks no transactions are being processed.

B.1.3.3 Proof-of-Elapsed Time

PoET is designed to address the inefficiency of PoW and replaces it with a protocol that is based on *trusted hardware*. A node that uses trusted hardware however, can be checked for certain properties such as whether it is running a certain software. This aids in relaxing the trust model in settings where the Byzantine's Generals Problem might be present [72].

Sawtooth Lake, a project by Hyperledger, leverages Intel's *Software Guard Extensions* (SGX) to establish a validation lottery that makes use of their CPUs capability to render a timestamp that is cryptographically signed by the hardware [115]. The PoET consensus protocol used for Hyperledger Sawtooth Lake is carried out in the following steps;

1. *Requesting wait time.* Every potential validator node request a secure *waiting time* from their hardware *enclave* that distributes these waiting times randomly. Then every node waits accordingly [177, 278]. The trusted hardware can afterwards produce certificates that indicate how much time has expired since the timer has started [72].

2. *Election of Validation Leader* After the assigned time has passed all nodes declare themselves to be validation leaders. Whichever node has the shortest wait time for a particular transaction block is elected the leader [72, 115]. More specifically, the node includes its PoET timestamp in the block, and if its waiting time is smaller than that generated by other nodes then the block is accepted [72]. Because each node has an equal chance of being chosen via their trusted hardware, the probability for a single entity of controlling the validation leadership is proportional to the resources contributed to the overall network [44, 177].

The steps that follow are similar to that of PoW; the elected leader creates a block and broadcasts it, thereafter the other miners validate the block. The PoET protocol can be regarded as more environmental friendly. However, the probability of becoming validation leader is proportional to the number of trusted hardware modules, and therefore economic investments still enhances one's influence on the consensus protocol. Furthermore, the security of the protocol is dependent on the hardware that could be running on a potential malicious host. The SGX for instance, is susceptible to *rollback attacks* [38] wherein a malicious user provides stale data to the trusted hardware's enclave and *key extraction attacks* [49] where an attacker leverages a vulnerability to extract attestation keys from SGX processors enabling a full break of the SGX [44].

B.1.3.4 Proof-of-Stake

As a response to the limitations of PoW the BCT community has turned towards Proof-of-Stake (PoS). The PoS consensus protocol has been introduced for public settings [183] with the aim to safeguard against Sybil attacks and malicious behavior by untrusted nodes [72]. The PoS protocol offers a more efficient and environmental friendly alternative to PoW as computing power is partially substituted by virtual resources (e.g. cryptocurrencies) that miners must invest to propose blocks [88, 157, 244, 282]. Rather than using computer power as a scarce resource to generate security, Proof of Stake uses the scarcity of the coin itself. Therefore nodes that participate in a PoS consensus protocol are more commonly referred to as *forgers* instead of miners [21, 158].

The idea behind the PoS model is that the more assets (e.g. cryptocurrency), or *stake* a node has, its incentive to undermine the system diminishes because subverting the system would inherently mean that the worth of the nodes' stake would decrease [177, 278]. Logically, this implies that one cannot participate in the consensus protocol without owning a stake [88]. A shared commonality of all PoS variants is that nodes that have more stake have a higher chance of generating new blocks [157, 183, 278, 285]. In other words, the more skin a forger puts in the game the higher its reward will be.

Coin age based PoS is one of the earliest implementations of the PoS consensus protocol used by PeerCoin. *Coin age* means that besides its market value a currency has an age [183, 285]. The accumulated *time* that a node holds his stake (currency) before using them to generate a block [158]. When used to generate a block the age of a coin is reset to 0. Only after a pre-determined time the coin can be used again as a stake to create a new block. The coin age system introduces the problem of *coin hoarding* which means that nodes do not spend their coins because they incentivized to hold on to them [183]. Rather than punishing inactive nodes, the Proof of Activity (PoA) protocol rewards stakeholders that contribute in sustaining the network. For stakeholders to collect a block reward they need to spend active online time on the network [30].

Cryptocurrencies like Blackcoin and NXT use randomization to predict the next forger of a block and do not rely on a mining process to create blocks. For these *chain-based PoS variants* an algorithm will review all users that have a stake and make a selection of the validating nodes based on the nodes' stake ratio compared to the entire system [278]. Although both variant of PoS to a degree succeed in decreasing energy waste, lowering latency and increasing throughput, Li et al. [158] suggest that both coin age based and the chain based variant of PoS have three security problems:

The first problem that corrodes the security is the *nothing at stake problem*. This attack can be carried out because they only need to have a stake, but not to commit a partition of the stake for the sake of forging a block. A second problem is a *stake grinding attack* where an attacker tries to bias the randomness of the election to forge a new block in their favor. This attack can be mounted in two ways; Either the attacker can "grind" through many combinations of parameters and find favorable parameters that would increase the probability of their coins generating a valid block. Or, the attacker can skip an opportunity to create a block to provide the forger a greater opportunity to forge future blocks in next rounds. The third problem is the *long range problem*, sometimes referred to as a history attack. Some blockchain platforms secure their chains for long range attacks by implementing *checkpoints*. A checkpoint is a block after which all prior chained blocks are regarded as final and immutable and are hard-coded in the system's software [158, 173]. Snow White based upon the work of Pass and Shi [204] designed to maintain robustness even if nodes sporadically participate in the consensus protocol. Ouroboros protocol [130], proposes a novel rewarding mechanism with the aim of increasing the incentive for nodes to behave honestly and to diminish the possibility of grinding attacks.

Byzantine fault tolerance PoS is a hybrid consensus protocol that combines the use of stakes (PoS) and aspects of the PBFT consensus protocol [274]. For instance, Tendermint adds a multi round voting system which makes the consensus protocol more complex. This PoS variant allows all staked nodes to participate in the block selection process [278]:

1. *Propose*. An algorithm embedded in the blockchain platform will randomly select several nodes with a stake to propose a block in a round-robin fashion.
2. *Prevote*. All nodes with a stake are asked to cast their vote for the next block. Only when 2/3 of all nodes vote to confirm the block can it continue to the precommit step.
3. *Precommit*. In the Precommit stage nodes engage in another round of voting for which yet again 2/3 of the nodes need to confirm the block to proceed to the next step.
4. *Commit* Once all votes have been casted and the block has been accepted it will be appended to the blockchain. A reward will be provided to the proposer of the block.

To overcome the nothing at stake problem the forgers are required to submit a *deposit* in the form of a stake to validate blocks. If forgers on purpose, or otherwise vote for blocks that contain invalid transactions their stake is slashed away and destroyed [72, 115]. Deposit based blockchain platforms have the problem that they lock down currency and thus decrease the liquidity of the network's currency. Furthermore, there could be an incentive to propagate conflicting blocks in order to double spend if the value of these transactions surpass the value of the deposit made by the forger [158].

According to literature there is also some general fundamental critique on the design principles of PoS consensus protocols. Firstly, nodes with a higher stake have more chance of mining a block [183, 278]. Furthermore, in PoS less energy is wasted and throughput and latency benefits are gained as compared to PoW. Yet, in essence PoS did not completely solve the problem of resource wasting, concentration of hashing power besides low throughput,

and high latency. However, as the mining costs are low attacks might come as a consequence which can reduce the security [183].

B.1.3.5 Delegated-Proof-of-Stake

(DPoS) Delegated Proof-of-Stake introduces another variant of PoS [183, 285]. In DPoS stakeholders elect delegates, referred to as *witnesses* to forge and validate blocks in round-robin fashion [158]. The consensus protocol of BitShares is an example of DPoS that is divided into two parts: electing a group of block producers and scheduling production;

1. *Vote for Witness*. Each stakeholder can select a witness based on their stake. The top number of the most selected witnesses that have participated in the voting round gain the authority to forge a block.
2. Thereafter the elected witnesses sequentially forge new blocks. A prerequisite is that the witness spends sufficiently enough time online. In the case a witness is unable to forge its assigned block stakeholders will vote for a new witness to substitute the defaulting witness. Accordingly, the activities for that block will be moved to the next [183].

Compared to PoW and Pos, DPoS is more energy efficient. Further, because the voting about the validity of a block is delegated and fewer nodes are needed to validate the blocks can be confirmed more quickly. Hence, as compared to PoW and PoS, DPoS has a low latency. Moreover, parameters including block size and block intervals can be adjusted by *committee members* of the governance board. When a delegate acts malicious this dishonest delegate can be voted out by all the other nodes [158, 285].

B.1.3.6 Zero-Knowledge-Proofs

Recently, different Zero-Knowledge-Proofs (ZKP's) based BCT networks have been proposed to preserve users' anonymity and confidentiality of transactions [274]. In general, ZKP's aim to confirm a statement about a transaction such as "This is a valid transaction" without revealing anything about the transfer (statement) itself or the parties involved [112, 274, 282].

Zerocoin was the first initiative with the aim of providing transaction unlinkability using ZKP's [72]. Similar to the Bitcoin Zerocoin uses the PoW consensus protocol to validate transactions. A *cryptographic mixer* is implemented for Zerocoin to conceal the links between a zerocoin and the corresponding Bitcoin. Mixing services group multiple transactions in such a way that a payment contains several input addresses and several output addresses [274]. When validating the transaction the miners do not have to verify the transaction with a digital signature, but rather validate whether coins belong to a list of valid coins [183]. Transaction unlinkability is achieved because the coin exchanged by the zerocoin owner can be any of the listed zerocoins [72, 274]. A series of mixing service can be linked to one another to further improve transaction unlinkability. If the mixed transactions are equal in value this further minimizes the traceability between the input and output addresses [274]. There are *centralized mixing services* that require a third party to enact the mixing, and *distributed mixing services* that do not require a third party to mix the coins. Coinsuffle [219] for instance, is a decentralized mixing protocol that is operated without any third party incurring only small additional overhead. Despite the fact that the cryptographic mixer unlinks the origins of the transactions to prevent transaction graph analysis the transactions' destination and amounts are still traceable [183].

Building on the ZKP approach as a foundation, Zcash, extent the privacy guarantees, and improve the efficiency (throughput and latency) of Zerocoin. Zcash uses a variant of the

PoW called Equihash. Transactions made using Zcash, including the split and merge transactions, are fully private [72]. Zcash employs a technique called *Zero-Knowledge-Succinct Non-Interactive Argument of Knowledge* (zk-SNARKS) to provide these privacy guarantees [183, 282] that are a specific type of ZKP. The transactions are based on a complex concept of ZKP that reveal only that unredeemed coins exist the sum of which is a certain value other information with regard to the origin and destination address will remain concealed [259]. Blockchains that implement zk-SNARK techniques incur large overheads due to the fact that ZKP's require public parameters that when stored can amount up to hundreds of megabytes [53, 72].

B.1.4 Physical View

The physical view is concerned with the topology of software components and their physical connections. Electronic devices known as nodes constitute a blockchains' P2P network and are the only physical connection to the non-digital world. P2P networks on which blockchain platforms are run have different arrangements; First, the network can be categorized on the basis of permissions (authorization). Second, networks can be categorized with regard to their accessibility. Permissions to perform operations on the blockchain might differ ranging from allowing anyone to read, write and to partake in the consensus protocol to only one of these permissions. Control over these permissions can be confined to a distinct group of nodes, or all nodes.

B.1.4.1 Permissionless Blockchain Models

As the name suggest *Permissionless* grant permission to all nodes in the P2P network to read and write transactions. By design permissionless blockchains are highly decentralized platforms, and permissions for all nodes are equal. These permissions were pre-defined when designing the blockchains architecture [272, 278]. Granting write and read permission to all nodes that participate in the P2P network allows anyone to contribute data to the distributed ledger, and obtain an identical copy of the ledger. It assures *censorship resistance*, preventing that an actor can block a valid transaction from being added to the ledger [88, 173, 261]. Furthermore, since multiple nodes hold a copy of the ledger and are permitted to read the transaction data, these systems are highly *transparent* [285]. Permissionless blockchains are designed in this fashion to ensure that all participants in the network have a *consistent view* of the distributed ledger [112]. The transparency of permissionless blockchains has a trade-off: It inherently means that individual nodes have less privacy with regards to their transactions. Currently most permissionless systems offer only *pseudo anonymity*, which means that the individual users are anonymous but as their transactions are transparent their accounts are not [62, 189, 244, 259]. Indeed, the origins of a transaction and their destination can be revealed [33, 254], and some research even lay bare behavior of Bitcoin users, how they spend their Bitcoins and the balance of Bitcoins they keep in their account.

However, permissionless blockchains ensures that transactions once recorded on their ledgers become nearly *immutable* [173]. Immutability means that transactions once recorded on the distributed ledger can not be erased or changed [88, 243] and only novel transactions can be appended [278]. A major drawback of permissionless systems is that in the event that security issues arise, or unauthorized transactions are executed, the response in the form of an update might be slow due to the hampered decision making [173].

B.1.4.2 Permissioned Blockchain Models

Permissioned blockchain platforms have confined and idiosyncratic permissions for their nodes [181, 231, 274, 278]. As such permissioned blockchain platforms offer high utility for organizations that require control over important functions. However, privacy of transactions is at the expense of immutability guarantees because confining the ability of every node to write transactions creates censorship resistance, whereas limiting the read ability of nodes in the network compromises transparency [173]. Since this model also requires a single entity that regulates permissions that could become a potential target of a cyberattack itself [189]. Another advantage of permissioned models is that updating the blockchain can be delegated to a select group of nodes. This reduces the time it will take to implement the update [44, 277]. In the event of any malfunctioning of the system the response time will therefore be significantly lower as compared to permissionless systems where multiple nodes need to come to an agreement.

The P2P network can also be described from the perspective of network accessibility. In the literature three categories of P2P networks can be distinguished that are coupled to a permission model introduced in the previous paragraph [44, 112, 143, 173, 274, 285].

B.1.4.3 Public Blockchain Networks

A *Public blockchain*, like the Bitcoin or Ethereum have *open network access* meaning that anyone willing is allowed to join the network. Public blockchains have no single owner. In fact they can not be owned by one single entity [193, 278]. Therefore, these networks can be regarded as *decentralized* [285]. Since the nodes do not know one another, they cannot trust each other [44, 244, 274]. This assumed *trustless* setting has important ramifications for the design of a public blockchains architecture [173]. For instance, Bitcoin was designed to assume that any node participating in the network needs to be distrusted, but the system will be secure for as long as the majority of nodes act honestly during the consensus protocol [78]. The Bitcoin and other public blockchains have to employ a complex and computational expensive consensus protocol to incentivize and regulate cooperative behavior between nodes [44, 71, 173]. Permissionless models are the hallmark of public blockchains [231]. The combination of open network access and a permissionless systems permits nodes that are part of the P2P network to perform all operations available. Consequently, for public blockchains transparency of transactions on the distributed ledger is paramount, and is as such embedded in the blockchains architecture [112].

B.1.4.4 Private Blockchain Networks

Private blockchains are blockchains networks that are owned by one organization. Blockchain networks that have one or few owners, and for which the permissions of the blockchain are vetted by a confined group of nodes can be regarded as more *centralized* [285]. These type of blockchains have limited network access which allows access only for authenticated nodes [71, 173, 231]. One gatekeeper, or a selection of nodes might have the permission to grant access to the network [112]. Another option is that only predefined nodes that are white listed can enter the network [53, 72, 272]. In essence, both options entail that before entering the network nodes are *authenticated* [71, 231].

Private blockchain platforms require that an administrator, or administrators assign permissions to each unique node [278]. The chances of nodes acting malicious are mitigated as their identity is known by the other nodes [173]. This does not imply however, that the nodes in the private blockchain network can fully trust each other [72]. Because the identities of the

nodes are known, the risks of Sybil attacks are slim. Computational intensive (thus expensive) consensus algorithms such as PoW or PoS are therefore less needed for the system to become byzantine fault tolerant [53, 71, 189].

B.1.4.5 Consortium Blockchain Networks

Consortium blockchains are similar to private blockchains in the sense that nodes first need to be authenticated before granted access to the network. However, consortium blockchains allow nodes from different organizations to access the blockchain network [183, 211, 274]. In addition, a consortium blockchain is usually owned by more than one entity. As such the degree of openness and centralization of consortium blockchains lies between a private and public blockchain [183]. Within consortium blockchain P2P networks the consensus process is confined to specific participants [209, 274, 285]. The right to read transactions stored on the blockchain network can be made public within the network, or restricted to a selected group of nodes [173, 274]. Consortium blockchains can therefore be perceived as being *partially decentralized*. Following similar reasoning as for private blockchain, the need for computationally expensive consensus algorithms is less present.

Appendix C

Example lease agreement

Lease Agreement

This Lease is made on 19th of July, 2021 between Bob Book, (hereafter called the Landlord) and John Doe, (hereafter called the Tenant).

1. PROPERTY:

- 1.1. Tenant agrees to rent from the Landlord and Landlord agrees to rent to the Tenant Clockstreet 12A, City of Chicago, State of Michigan hereafter referred to as the Premises.

2. TERM:

- 2.1. The term of this lease is: of 2 Years starting on 1 January, 2021 and ending on 1 January, 2023.
- 2.2. The Landlord is not responsible if the Landlord cannot give the Tenant possession of the Premises at the start of this Lease.
- 2.3. However, rent will only be charged from the date on which possession of the Premises is made available to the Tenant.
- 2.4. If the Landlord cannot give possession within 30 days after the starting date, the Tenant may cancel this Lease.

3. RENT:

- 3.1. The rent of the Premises will be \$1000.
- 3.2. The Tenant will pay the rent, in advance, on the 1st day of each month.
- 3.3. The first payment of rent and any security deposit is due by 01 January, 2021 prior to moving in.
- 3.4. The Tenant must pay a late charge of \$500 for each payment that is more than 30 days late.
- 3.5. This charge is due with and shall be considered to be a part of the monthly rent payment for the month in which the rent was paid late.

4. SECURITY DEPOSIT:

- 4.1. The Tenant will deposit the sum of \$2000 with the Landlord as security that the Tenant will comply with all the terms of this Lease.
- 4.2. This money is being held by the Landlord in a tenant security deposit account at ADDRESS.
- 4.3. If the Tenant complies with the terms of this Lease, the Landlord will return this deposit within 30 days after the end of the Lease.
- 4.4. The Landlord may use as much of the security deposit as necessary to pay for damages resulting from the Tenant's occupancy or, at Landlord's sole option and election, to pay for delinquent or unpaid rent and late charges.

- 4.5. If the Landlord uses the deposit for such mentioned instances prior to the Lease termination, the Landlord may demand that the Tenant replace the amount of the security deposit used by the Landlord.
- 4.6. If the Landlord sells the property, the Landlord may transfer the deposit to the new owners for the Tenant's benefit.
- 4.7. If the Landlord sells the property and transfers the deposit to the new owners, The Landlord will notify the Tenant of any sale and transfer of the deposit.
- 4.8. The Landlord will thereupon be released of all liability to return the security deposit.
5. LANDLORD'S AGENT:
 - 5.1. The Landlord authorizes the following person(s) to manage the Premises on behalf of the Landlord: The Cleaning Company.
6. USE OF THE PREMISES:
 - 6.1. The Tenant may use the Premises only as a single-family residence.
7. UTILITIES:
 - 7.1. The Landlord will pay for the following utilities: Garbage Removal, Gas, and Oil.
 - 7.2. The Tenant will pay for the following utilities: Water, Sewer, and Electricity.
8. EVICTION:
 - 8.1. If the Tenant does not pay the rent within 60 days of the date when it is due, the Tenant may be evicted.
 - 8.2. The Landlord may also evict the Tenant if the Tenant does not comply with all of the terms of this Lease, or for any other causes allowed by law.
 - 8.3. If evicted, the Tenant must continue to pay the rent during the remainder of the term.
 - 8.4. The Tenant must also pay all costs, including reasonable attorney fees, related to the eviction and the collection of any monies owed to the Landlord, along with the cost of re-entering, re-renting, cleaning and repairing the Premises.
 - 8.5. Rent received from any new tenant during the remaining term of this lease will be applied by the Landlord to reduce rent only, which may be owed by the Tenant.
9. PAYMENTS BY LANDLORD:
 - 9.1. If the Tenant fails to comply with the terms of this Lease, the Landlord may take any required action and charge the cost, including reasonable attorney fees, to the Tenant.
 - 9.2. Failure to pay such costs upon demand is a violation of this Lease.
10. CARE OF THE PREMISES:
 - 10.1. The Tenant has examined the Premises, including (where applicable) the living quarters, all facilities, furniture and appliances, and is satisfied with its present physical condition.
 - 10.2. The Tenant agrees to maintain the Premises in as good condition as it is at the start of this Lease except for ordinary wear and tear.
 - 10.3. The Tenant must pay for all repairs, replacements, and damages, whether or not caused by the act or neglect of the Tenant.
 - 10.4. The Tenant will remove all of the Tenant's property at the end of this Lease.
 - 10.5. Any property that is left becomes the property of the Landlord and may be thrown out.
 - 10.6. All of Tenant's garbage will be disposed of properly by Tenant in the appropriate receptacles for garbage collection.
 - 10.7. Accumulations of garbage in and around the Premises or depositing by Tenant or those residing with Tenant of garbage in areas not designated and designed as garbage receptacles shall constitute a violation of this lease.
 - 10.8. Tenant shall generally maintain the Premises in a neat and orderly condition.

- 10.9. Damage or destruction by Tenant, Tenant's employees or Tenant's visitors of the Premises shall constitute a violation of this Lease.
11. DESTRUCTION OF Premises:
- 11.1. If the Premises are destroyed through no fault of the Tenant, the Tenant's employees or Tenant's visitors, then the Lease will end, and the Tenant will pay rent up to the date of destruction.
12. INTERRUPTION OF SERVICES:
- 12.1. The Landlord is not responsible for any inconvenience or interruption of services due to repairs, improvements or for any reason beyond the Landlord's control.
13. ALTERATIONS: The Tenant must get the Landlord's prior written consent to alter, improve, paint or wallpaper the Premises.
- 13.1. Alterations, additions, and improvements become the Landlord's property.
14. COMPLIANCE WITH LAWS:
- 14.1. The Tenant must comply with laws, orders, rules, and requirements of governmental authorities and insurance companies which have issued or are about to issue policies covering the Premises and/or its contents.
15. NO WAIVER BY LANDLORD:
- 15.1. The Landlord does not give up or waive any rights by accepting rent or by failing to enforce any terms of this Lease.
16. NO ASSIGNMENT OR SUBLEASE:
- 16.1. The Tenant may not sublease the Premises or assign this Lease without the Landlord's prior written consent.
17. ENTRY BY LANDLORD:
- 17.1. Upon reasonable notice, the Landlord may enter the Premises to provide services, inspect, repair, improve or show the Premises.
- 17.2. The Tenant must notify the Landlord if the Tenant is away for 30 days or more.
- 17.3. In case of an emergency or the Tenant's absence, the Landlord may enter the Premises without the Tenant's consent.
18. QUIET ENJOYMENT:
- 18.1. The Tenant may live in and use the Premises without interference subject to the terms of this Lease.
19. SUBORDINATION:
- 19.1. This Lease and the Tenant's rights are subject and subordinate to present and future mortgages on the property which include the Premises.
- 19.2. The Landlord may execute any papers on the Tenant's behalf as the Tenant's attorney in fact to accomplish this.
20. HAZARDOUS USE:
- 20.1. The Tenant will not keep anything in the Premises which is dangerous, flammable, explosive or which might increase the danger of fire or any other hazard, or which would increase Landlord's fire or hazard insurance.
21. INJURY OR DAMAGE:
- 21.1. The Tenant will be responsible for any injury or damage caused by the act or neglect of the Tenant, the Tenant's employees or Tenant's visitors.
- 21.2. The Landlord is not responsible for any injury or damage unless due to the negligence or improper conduct of the Landlord.

22. RENEWALS AND CHANGES IN LEASE:

- 22.1. Upon expiration of the rental term provided for above, this lease shall automatically renew itself, indefinitely, for successive one-month periods, unless modified by the parties.
- 22.2. The Landlord may modify this lease or offer the Tenant a new lease by forwarding to the Tenant a copy of the proposed changes or a copy of the new lease.
- 22.3. If changes in this lease or a new lease are offered, the Tenant must notify the Landlord of the Tenant's decision to stay within thirty (30) days of the date the proposed changes or the copy of the new lease is received by the Tenant.
- 22.4. If the Tenant fails to accept the lease changes or the new lease within thirty (30) days of the date the proposed changes or new lease is offered, the Tenant may be evicted by the Landlord, as provided for in State law.
- 22.5. Nevertheless, if the rent is increased by the lease changes or new lease, the Tenant will be obligated to pay the new rent, regardless of whether the Tenant has affirmatively accepted the lease changes or new lease, if the Tenant continues to occupy the property on the date the new rent becomes effective.

23. PETS:

- 23.1. No dogs, cats, or other animals are allowed on the Premises without the Landlord's prior written consent.

24. NOTICES:

- 24.1. All notices provided by this Lease must be written and delivered personally or by certified mail, return receipt requested, to the parties at their addresses listed above, or to such other address as the parties may from time to time designate.
- 24.2. Notices to the Landlord must also be sent to the Landlord's agent listed above (if any).

25. SIGNS:

- 25.1. The Tenant may not put any sign or projection (such as a T.V. or radio antenna) in or out of the windows or exteriors of the Premises without the Landlord's prior written consent.

26. HOLDOVER RENT:

- 26.1. Should this Lease be terminated, either through a valid notice of dispossession by the Landlord, or through order of a court, and should Tenant remain on the Premises thereafter, then Tenant shall be liable to pay rent at a rate of double the base rent provided for under this lease, from the date of termination until such time as Tenant vacates the Premises, whether Tenant vacates the Premises voluntarily or through enforcement of an order for eviction.

27. VALIDITY OF LEASE:

- 27.1. If a clause or provision of this Lease is legally invalid, the rest of this Lease remains in effect.
- 27.2. If a clause or provision of this lease is ambiguous, and it may be interpreted in a manner either consistent or inconsistent with existing law, it shall be interpreted in a manner consistent with existing law.

28. PARTIES:

- 28.1. The Landlord and each of the Tenant(s) are bound by this Lease.
- 28.2. All parties who lawfully succeed to their rights and responsibilities are also bound.

29. GENDER:

- 29.1. The use of any particular gender (masculine, feminine or neuter) and case (singular or plural) in this Lease is for convenience, only.

29.2. No inference is to be drawn therefrom.

29.3. The correct gender and case is to be freely substituted throughout, as appropriate.

30. TENANT'S ACKNOWLEDGMENT:

30.1. The Tenant acknowledges having read all of the terms and conditions of this lease and the attached rules and regulations.

30.2. Tenant acknowledges that no oral representations have been made to him by the Landlord or the Landlord's agent(s) other than the representations contained in this Lease.

30.3. The Tenant acknowledges that he/she is relying only upon the promises and representations contained in this Lease.

31. ENTIRE LEASE:

31.1. All promises the Landlord has made are contained in this written Lease.

31.2. This Lease can only be changed by an agreement in writing by both the Tenant and the Landlord.

32. SIGNATURES:

32.1. The Landlord and the Tenant agree to the terms of this Lease.

32.2. If this Lease is made by a corporation, its proper corporate officers sign and its corporate seal is affixed.

Appendix D

Instantiation of Motivating Example

```

1  {
2    "contractName": "ContractExample",
3    "contractId": 1,
4    "contractModel": [
5      {
6        "party": {
7          "partyId": "Party1",
8          "partyName": "Bob Book",
9          "entityType": "natural"
10       }
11     },
12     {
13       "party": {
14         "partyId": "Party2",
15         "partyName": "John Doe",
16         "entityType": "natural"
17       }
18     },
19     {
20       "definition": {
21         "definitionId": "Definition1",
22         "definition": [
23           {
24             "definiens": {
25               "definiensName": "the Landlord",
26               "definiensId": "Definiens1"
27             }
28           },
29           {
30             "defiendum": [
31               {
32                 "party": {
33                   "partyId": "Party1",
34                   "partyName": "Bob Book",
35                   "entityType": "natural"
36                 }
37             ]
38           }
39         ]
40       }
41     },
42     {
43       "definition": {
44         "definitionId": "Definition2",
45         "definition": [
46           {
47             "definiens": {
48               "definiensName": "the Tenant",
49               "definiensId": "Definiens2"
50             }
51           },
52           {
53             "defiendum": [

```

```

55         {
56             "party": {
57                 "partyId": "Party2",
58                 "partyName": "John Doe",
59                 "entityType": "natural"
60             }
61         }
62     ]
63 }
64 ]
65 }
66 },
67 {
68     "definition": {
69         "definitionId": "Definition3",
70         "definition": [
71             {
72                 "definiens": {
73                     "definiensName": "the term of the lease",
74                     "definiensId": "Definiens3"
75                 }
76             },
77             {
78                 "defiendum": [
79                     {
80                         "term": {
81                             "termId": "Term1",
82                             "type": "span",
83                             "timeexpression": {
84                                 "timeExpressionId": "TimeExpression1",
85                                 "type": "on"
86                             },
87                             "timePoint": {
88                                 "timeSpan": {
89                                     "timeSpanId": "TimeSpan1",
90                                     "start": {
91                                         "date": {
92                                             "day": 1,
93                                             "month": 1,
94                                             "year": 2021
95                                         }
96                                     },
97                                     "end": {
98                                         "date": {
99                                             "day": 1,
100                                             "month": 1,
101                                             "year": 2023
102                                         }
103                                     }
104                                 }
105                             }
106                         }
107                     }
108                 ]
109             }
110         ]
111     },
112     {
113         "definition": {
114             "definitionId": "Definition4",
115             "definition": [
116                 {
117                     "definiens": {
118                         "definiensName": "the rent",
119                         "definiensId": "Definiens4"
120                     }
121                 },
122                 {
123                     "defiendum": [
124                         {
125                             "resource": {
126

```



```

127         "resourceId": "Resource1",
128         "resourceName": "$",
129         "quantity": {
130             "quantName": "the",
131             "quantId": "Quantity1",
132             "value": 500
133         }
134     }
135 }
136 ]
137 }
138 ]
139 }
140 },
141 {
142     "definition": {
143         "definitionId": "Definition5",
144         "definition": [
145             {
146                 "definiens": {
147                     "definiensName": "the security deposit",
148                     "definiensId": "Definiens5"
149                 }
150             },
151             {
152                 "defiendum": [
153                     {
154                         "resource": {
155                             "resourceId": "Resource2",
156                             "resourceName": "security deposit",
157                             "quantity": {
158                                 "quantName": "the",
159                                 "quantId": "Quantity2",
160                                 "value": 2000
161                             }
162                         }
163                     }
164                 ]
165             }
166         ]
167     }
168 },
169 {
170     "definition": {
171         "definitionId": "Definition6",
172         "definition": [
173             {
174                 "definiens": {
175                     "definiensName": "a late charge",
176                     "definiensId": "Definiens6"
177                 }
178             },
179             {
180                 "defiendum": [
181                     {
182                         "resource": {
183                             "resourceId": "Resource3",
184                             "resourceName": "$",
185                             "quantity": {
186                                 "quantName": "a",
187                                 "quantId": "Quantity3",
188                                 "value": 500
189                             }
190                         }
191                     }
192                 ]
193             }
194         ]
195     }
196 },
197 {
198     "clause": {

```

```

199     "clauseId": 1,
200     "clauseType": "Normal",
201     "TriggeringEvent": {
202         "triggerId": "Trigger1",
203         "Contains": [
204             {
205                 "event": {
206                     "eventName": "start lease",
207                     "eventId": "Event1",
208                     "eventType": "temporal"
209                 }
210             }
211         ],
212         "Conditions": [
213             {
214                 "condition": {
215                     "conditionId": "Condition1",
216                     "Contains": [
217                         {
218                             "action": {
219                                 "actionId": "Action1",
220                                 "name": "pay the Rent",
221                                 "modal": "OBLIGATION",
222                                 "group": "Pay",
223                                 "repeated": false,
224                                 "status": "unfinished",
225                                 "resource": {
226                                     "resourceId": "Resource4",
227                                     "resourceName": "rent",
228                                     "definedAs": "Definition4",
229                                     "quantity": {
230                                         "quantName": "the",
231                                         "quantId": "Quantity4",
232                                         "value": 500,
233                                         "definedAs": "Definition4"
234                                     }
235                                 }
236                             },
237                             "promissor": {
238                                 "promissorName": "the tenant",
239                                 "promissorId": "Promissor1",
240                                 "party": "Party1"
241                             },
242                             "promissee": {
243                                 "promisseeName": "the landlord",
244                                 "promisseeId": "Promissee1",
245                                 "party": "Party2"
246                             }
247                         },
248                         "term": {
249                             "termId": "Term3",
250                             "type": "span",
251                             "timeexpression": {
252                                 "timeExpressionId": "
253                                     TimeExpression2",
254                                 "type": "on"
255                             }
256                         },
257                         "timePoint": {
258                             "timeSpan": {
259                                 "timeSpanId": "TimeSpan2",
260                                 "start": {
261                                     "date": {
262                                         "day": 1,
263                                         "month": 1,
264                                         "year": 2021
265                                     }
266                                 },
267                                 "end": {
268                                     "date": {
269                                         "day": 1,
270                                         "month": 1,
271                                         "year": 2023
272                                     }
273                                 }
274                             }
275                         }
276                     ]
277                 }
278             }
279         ]
280     }
281 }

```

```

271     }
272     },
273     "repeatedEach": {
274       "timeUnitId": "TimeUnit1",
275       "type": "month",
276       "quantity": {
277         "quantName": "1",
278         "quantId": "Quantity5",
279         "value": 1
280       }
281     }
282   }
283 },
284 "Remedies": [
285   {
286     "remedy": {
287       "remedyId": "Remedy1",
288       "Contains": [
289         {
290           "action": {
291             "actionId": "Action2",
292             "name": "pay a late charge",
293             "modal": "OBLIGATION",
294             "group": "Pay",
295             "repeated": false,
296             "status": "unfinished",
297             "resource": {
298               "resourceId": "Resource5",
299               "resourceName": "late charge",
300               "definedAs": "Definition6",
301               "quantity": {
302                 "quantName": "the",
303                 "quantId": "Quantity6",
304                 "value": 500,
305                 "definedAs": "Definition6"
306               }
307             }
308           },
309           "promissor": {
310             "promissorName": "the tenant",
311             "promissorId": "Promissor2",
312             "party": "Party1"
313           },
314           "promisee": {
315             "promiseeName": "the landlord",
316             "promiseeId": "Promisee2",
317             "party": "Party2"
318           },
319           "term": {
320             "termId": "Term4",
321             "type": "relative",
322             "timeexpression": {
323               "timeExpressionId": "TimeExpression4",
324               "type": "after"
325             }
326           },
327           "timepoint": {
328             "relativeTimePoint": {
329               "relatTimePointId": "relatTimePoint1",
330               "relatedTo": "Action1",
331               "timeUnit": {
332                 "timeUnitId": "timeUnit2",
333                 "type": "day",
334                 "quantity": {
335                   "quantName": "30"
336                 }
337               }
338             }
339           }
340         }
341       ]
342     }
343   }
344 ]

```

```

335                                     "quantId": "
336                                             Quantity7
337                                             ",
338                                             "value": 30
339                                     }
340                                 }
341                            }
342                        }
343                    }
344                ]
345            }
346        },
347        {
348            "remedy": {
349                "remedyId": "Remedy2",
350                "Contains": [
351                    {
352                        "action": {
353                            "actionId": "Action3",
354                            "name": "evict tenant",
355                            "modal": "PERMISSION",
356                            "group": "terminate",
357                            "repeated": false,
358                            "status": "unfinished",
359                            "resource": {
360                                "resourceId": "Resource6",
361                                "resourceName": "evict",
362                                "quantity": {
363                                    "quantName": "the",
364                                    "quantId": "Quantity8",
365                                    "value": 1
366                                }
367                            },
368                            "promissor": {
369                                "promissorName": "the landlord",
370                                "promissorId": "Promissor3",
371                                "party": "Party2"
372                            },
373                            "promisee": {
374                                "promiseeName": "the tenant",
375                                "promiseeId": "Promisee3",
376                                "party": "Party1"
377                            },
378                            "term": {
379                                "termId": "Term5",
380                                "type": "relative",
381                                "timeexpression": {
382                                    "timeExpressionId": "
383                                            TimeExpression5",
384                                    "type": "after"
385                                },
386                                "timepoint": {
387                                    "relativeTimePoint": {
388                                        "relatTimePointId": "
389                                            relatTimePoint2",
390                                        "relatedTo": "Action1",
391                                        "timeUnit": {
392                                            "timeUnitId": "
393                                                timeUnit3",
394                                            "type": "day",
395                                            "quantity": {
396                                                "quantName": "
397                                                    60",
398                                                "quantId": "
399                                                    Quantity9",
400                                                "value": 60
401                                            }
402                                        }
403                                    }
404                                }
405                            }
406                        }
407                    }
408                ]
409            }
410        }
411    ],
412    "type": "contract"
413 }

```

```

397     }
398   }
399   }
400   }
401   }
402   }
403   ]
404   }
405   }
406   ]
407   }
408 }
409 ],
410 "Promises": [
411   {
412     "promise": {
413       "promiseId": "Promise1",
414       "Contains": [
415         {
416           "action": {
417             "actionId": "Action4",
418             "name": "pay the security deposit",
419             "modal": "OBLIGATION",
420             "group": "deposit",
421             "repeated": false,
422             "status": "unfinished",
423             "resource": {
424               "resourceId": "Resource7",
425               "resourceName": "security deposit",
426               "definedAs": "Definition5",
427               "quantity": {
428                 "quantName": "the",
429                 "quantId": "Quantity10",
430                 "value": 2000,
431                 "definedAs": "Definition5"
432               }
433             },
434             "promissor": {
435               "promissorName": "the tenant",
436               "promissorId": "Promissor4",
437               "party": "Party1"
438             },
439             "promissee": {
440               "promisseeName": "the landlord",
441               "promisseeId": "Promissee4",
442               "party": "Party2"
443             },
444             "term": {
445               "termId": "Term2",
446               "type": "absolute",
447               "timeexpression": {
448                 "timeExpressionId": "TimeExpression2",
449                 "type": "before"
450               },
451               "timePoint": {
452                 "absoluteTimePoint": {
453                   "absolutePointId": "TimePoint1",
454                   "date": {
455                     "day": 1,
456                     "month": 1,
457                     "year": 2021
458                   }
459                 }
460             }
461           }
462         }
463       ]
464     }
465   },
466   {
467     "promise": {

```

```

469     "promiseId": "Promise2",
470     "Contains": [
471     {
472         "action": {
473             "actionId": "Action5",
474             "name": "pay for gas",
475             "modal": "OBLIGATION",
476             "group": "Pay",
477             "repeated": false,
478             "status": "unfinished",
479             "resource": {
480                 "resourceId": "Resource8",
481                 "resourceName": "gas",
482                 "quantity": {
483                     "quantName": "the",
484                     "quantId": "Quantity?",
485                     "value": "?"
486                 }
487             },
488             "promissor": {
489                 "promissorName": "the tenant",
490                 "promissorId": "Promissor5",
491                 "party": "Party1"
492             },
493             "promissee": {
494                 "promisseeName": "the landlord",
495                 "promisseeId": "Promissee5",
496                 "party": "Party2"
497             },
498             "term": {
499                 "termId": "Term6",
500                 "type": "span",
501                 "timeexpression": {
502                     "timeExpressionId": "TimeExpression6",
503                     "type": "during"
504                 },
505                 "timePoint": {
506                     "timeSpan": {
507                         "timeSpanId": "TimeSpan3",
508                         "start": {
509                             "date": {
510                                 "day": 1,
511                                 "month": 1,
512                                 "year": 2021
513                             }
514                         },
515                         "end": {
516                             "date": {
517                                 "day": 1,
518                                 "month": 1,
519                                 "year": 2023
520                             }
521                         }
522                     }
523                 }
524             }
525         }
526     },
527     {
528         "action": {
529             "actionId": "Action6",
530             "name": "pay for water",
531             "modal": "OBLIGATION",
532             "group": "Pay",
533             "repeated": false,
534             "status": "unfinished",
535             "resource": {
536                 "resourceId": "Resource9",
537                 "resourceName": "water",
538                 "definedAs": "Definition5",
539                 "quantity": {
540                     "quantName": "the",

```



```

613         "repeated": false,
614         "status": "unfinished",
615         "resource": {
616             "resourceId": "Resource9",
617             "resourceName": "the terms", //how to
618                 do this?//
619             "quantity": {
620                 "quantName": "the", //how to do
621                     this?//
622                 "quantId": "Quantity12",
623                 "value": 500
624             }
625         },
626         "promissor": {
627             "promissorName": "the tenant",
628             "promissorId": "Promissor6",
629             "party": "Party1"
630         },
631         "promisee": {
632             "promiseeName": "the landlord",
633             "promiseeId": "Promisee6",
634             "party": "Party2"
635         }
636     }
637 },
638 "promise": [
639     {
640         "promise": {
641             "promiseId": "Promise3",
642             "Contains": [
643                 {
644                     "action": {
645                         "actionId": "Action7",
646                         "name": "return this deposit",
647                         "modal": "OBLIGATION",
648                         "group": "Pay",
649                         "repeated": false,
650                         "status": "unfinished",
651                         "resource": {
652                             "resourceId": "Resource10",
653                             "resourceName": "deposit",
654                             "definedAs": "Definition5",
655                             "quantity": {
656                                 "quantName": "the",
657                                 "quantId": "Quantity13",
658                                 "value": 2000,
659                                 "definedAs": "Definition5"
660                             }
661                         }
662                     },
663                     "term": {
664                         "termId": "Term7",
665                         "type": "relative",
666                         "timeexpression": {
667                             "timeExpressionId": "
668                                 TimeExpression6",
669                             "type": "after"
670                         }
671                     },
672                     "timepoint": {
673                         "relativeTimePoint": {
674                             "relatTimePointId": "
675                                 relatTimePoint3",
676                             "relatedTo": "Trigger3
677                                 ",
678                             "timeUnit": {
679                                 "timeUnitId": "
680                                     timeUnit4",
681                                 "type": "day",
682                                 "quantity": {
683                                     "quantName": "
684                                         30",

```



```
677                                     "quantId": "
678                                     Quantity14
679                                     ",
680                                     "value": 30
681                                     }
682                                     }
683                                     }
684                                     }
685                                     }
686                                     ]
687                                     }
688                                     }
689                                     ]
690                                     }
691                                     }
692                                     ]}
693                                     }
694                                     }
695                                     ]
696                                     }
```

Listing D.1: Complete model of motivating example

Appendix E

Questionnaire and initial concepts Delphi method

E.1 Intro

Thank you for taking the time to participate in this study. For my Ph.D. at Tilburg University and the JADS I am conducting research on how smart contracts could be modelled. We would love to tap into your expertise about blockchains and in particular smart contracts.

With your help we would also like to better understand the key concepts used for smart contracts, and establish a model that captures these concepts using terminology that is used in practice.

Filling in the questionnaire for this first Delphi round should take about 5 minutes, and your responses are completely anonymous. Your responses will be used only for scientific purposes and will be deleted once the study is completed.

You can take the survey only once, but you can edit your responses until you have submitted the questionnaire.

If you have any questions about the survey, please feel free to contact me: b.j.butijn@jads.nl
Your input is really appreciated.

E.2 Instructions

Again, thank you for taking the time to participate in the Delphi study. Please read the instructions below before proceeding to answer any questions.

Hereafter a list is presented of concepts related to «PLATFORM NAME» smart contracts. We would like you to indicate for each concept if the concept is an alias to another concept. In a following question you can suggest to add missing concepts. In a next round these concepts will be presented to other panelists participating in the study.

Question 1: From various sources we have identified concepts related to smart contracts on the «PLATFORM NAME». Below you can find a list of these concepts with a description:

«LIST OF CONCEPTS»

Are there any concepts in the list that are unfamiliar to you?

1. Yes
2. No

Question 2: In the previous question you indicated that one or more concept were unfamiliar to you. Could you please specify which concept(s) are unfamiliar? You can add an unfamiliar concept by clicking the "Add Concept" button.

This concept is unfamiliar

Question 3: In your opinion, are there any concepts in the list that actually describe one and the same concept (are they an alias of one another)?

1. Yes
2. No

Question 4: Could you please indicate which concept is an alias to another concept? You can add an alias by clicking the "Add Alias" button.

| | | |
|----------------------------------|----------------------------------|--|
| This concept: | Is the same as this concept: | |
| <input type="text"/> | <input type="text"/> | <input type="button" value="Add Alias"/> |
| <input type="button" value="▼"/> | <input type="button" value="▼"/> | |

Question 5: Hereunder you will again find a list of the concepts we identified:

«LIST OF CONCEPTS»

In your opinion, are there any concepts related to «PLATFORM NAME» smart contracts missing?

1. Yes
2. No

Question 6: Here is again the complete list of the concepts.

«LIST OF CONCEPTS»

Could you please indicate which additional concepts are missing from the list? Please also provide a description of the concept. You can propose a concept by clicking the plus "Add Concept" button.

| | | |
|----------------------|----------------------|--|
| Concept: | Description: | |
| <input type="text"/> | <input type="text"/> | <input type="button" value="Add Concept"/> |

Appendix F

Enum Types Used in Models

This appendix depicts the enum types used for each respective (meta) model. The containers with the heading «Enum» are the different enum types and the boldface denotes the name of the enum. The words below the first bar under the name of the Enum type are the options for that enum.

F.1 Enums used for Ethereum smart contract metamodel

| | | | |
|--|---|--|--|
| <div><div><<Enum>></div><div>OperatorType</div></div> <div>Addition Subtraction Modulus Increment Decrement Division Multiplication</div> | <div><div><<Enum>></div><div>RelOperatorType</div></div> <div>greaterThan equalTo smallerThan notEqualTo</div> | <div><div><<Enum>></div><div>StructureType</div></div> <div>Variable Enum Array Mapping</div> | <div><div><<Enum>></div><div>GlobalType</div></div> <div>Now MessageSender MessageValue</div> |
| <div><div><<Enum>></div><div>PlatformType</div></div> <div>Ethereum Hyperledger</div> | <div><div><<Enum>></div><div>FunctionVisibility</div></div> <div>Private Public Internal External</div> | <div><div><<Enum>></div><div>InteractionType</div></div> <div>None Invokable Payable</div> | <div><div><<Enum>></div><div>VisibilityType</div></div> <div>Private Public Internal</div> |

Figure F.1: Enum types used for Ethereum smart contract metamodel

F.2 Enums used for Hyperledger smart contract metamodel

| <<Enum>> DataType | <<Enum>> InteractionType | <<Enum>> OperatorType | <<Enum>> GlobalType | | | |
|---|---|--|--|--|--|--|
| String | None | Addition | currentTime | | | |
| Boolean | Invokable | Subtraction | senderTransaction | | | |
| Datetime | <<Enum>> OperationType | Modulus | valueTransaction | | | |
| Double | | Increment | VarScope | | | |
| Byte | | Decrement | | | | |
| Integer | | Division | | | | |
| <<Enum>> RelOperatorType | Update | Multiplication | <<Enum>> PlatformType | | | |
| | Read | Ethereum | | | | |
| | <<Enum>> ActionType | | | | | |
| | | | | | | |
| greaterThan | Not | Hyperledger | | | | |
| equalTo | Allow | | | | | |
| smallerThan | | | | | | |
| notEqualTo | | | | | | |

Figure F.2: Enum types used for Hyperledger Fabric smart contract metamodel

F.3 Enums used for platform independent model

| | | | | |
|--|---|--|---|-------------|
| VarScope | <<Enum>> VisibilityType | <<Enum>> StructureType | <<Enum>> OperatorType | |
| Local | Private | Variable | Addition | |
| Global | Public | Enum | Subtraction | |
| <<Enum>> GlobalType | Internal | Array | Modulus | |
| | <<Enum>> RelOperatorType | KeyValue | Increment | |
| | | <<Enum>> PreConditionType | Decrement | |
| | | | Division | |
| Multiplication | | | | |
| <<Enum>> AssetType | currentTime | | <<Enum>> OperationType | |
| | senderTransaction | | | greaterThan |
| | valueTransaction | equalTo | | |
| <<Enum>> AssetType | smallerThan | <<Enum>> ActionType | <<Enum>> OperationType | |
| | notEqualTo | | | Read |
| | <<Enum>> InteractionType | | | |
| Unique | None | Read | | |
| Non-Unique | | Delete | | |
| <<Enum>> PlatformType | | Invokable | Update | |
| | Transfers | Create | | |
| | | | | |
| Ethereum | | | | |
| Hyperledger | | | | |

Figure F.3: Enum types used for PIM smart contract metamodel

Appendix G

Transformation Rules for Models

This appendix contains all transformation rules required to transform the platform independent model to one of the platform specific models. The appendix also contains all constraints required to check the PIM.

G.1 Transformation Rules From PIM to Ethereum PSM

```

1  helper def :setInteractionType(fu.interaction : String) :
    InteractionType =
2      switch (fu.interaction) {
3          case(None) : {InteractionType.None} break;
4          case(Invokable) : {InteractionType.Invokable} break;
5          case(Transfers) : {InteractionType.Payable} break;
6      }
7
8  helper def :setStructType(structType : String) : String =
9      switch (structType) {
10         case("Enum"):{StructureType.Enum} break;
11         case("Variable"):{StructureType.Variable} break;
12         case("Array"):{StructureType.Array} break;
13         case("KeyValue"):{StructureType.Mapping} break;
14     }
15
16  helper def :setValueType(val : String) : String =
17      switch (val) {
18         case("String"):{DataType.String} break;
19         case("Integer"):{DataType.Integer} break;
20         case("Byte"):{DataType.Byte} break;
21         case(Address):{DataType.Address} break;
22     }
23
24  helper def :setType(dType : String) : String =
25      switch (dType) {
26         case("String"):{"String"} break;
27         case("Integer"):{"Integer"} break;
28         case("Byte"):{"Byte"} break;
29         case(Address):{"Address"} break;
30     }
31
32  helper context solPSM!SmartContract def : getModifiers() :
    Sequence(solPSM!Modifier) =
33      self.Functions->collect(fu | fu.Modifiers)->
34      asSet()->collect(mod | thisModule.createModifier(mod));
35

```

```

36 helper def :setStructAttr(co : Sequence(scPIM!Concept)) :
      Sequence(solPSM!Struct) =
37     co.Contains->collect(str | thisModule.createStruct(str));
38
39 helper def :setStructConstruction(co : sequence(scPIM!Concept)) :
      Sequence(solPSM!LocalVariable) =
40     co.Attributes->collect(var | thisModule.createLocalVariable(var));

1 rule Meta2Meta {
2     from
3         me : scPIM!Meta
4     to
5         met : solPSM!Meta(
6             target<-me.target
7             nameSpace<-me.nameSpace
8             smartContracts<-me.smartContracts->collect(sc |
9                 thisModule.SC2SC(sc))
10            assets<-me.assets->collect(as | thisModule.Agent2solAgent(as))
11            agents<-me.agents->collect(ag | thisModule.Asset2solAsset(ag))
12        )
13
14 lazy rule Agent2solAgent {
15     from
16         ag : scPIM!Agent
17     to
18         age : solPSM!solAgent(
19             agentName<-ag.agentName
20             agentId<-ag.agentId
21             address<-ag.address
22         )
23 }
24
25 lazy rule Asset2solAsset {
26     from
27         as : scPIM!Asset
28     to
29         ass : solPSM!solAsset
30             assetName<-as.assetName
31             assetId<-as.assetId
32             amount<-as.amount
33             symbol<-as.symbol
34             assetType<-as.assetType
35             owners<-as.owners
36             initAmount<-as.initAmount
37         )
38 }
39
40 lazy rule SC2SC {
41     from
42         sc : scPIM!SmartContract
43     to
44         ssc : solPSM!SmartContract(
45             SCName<-sc.SmartContractName
46             abstract<-sc.abstract
47             owner<-sc.owner

```

```

48         Imports<-sc.Imports->collect(el | thisModule.createImport(el))
49         Enums<-sc.Enums->collect(el | thisModule.createEnum(el))
50         StateVariables<-sc.GlobalVariables->collect(el |
51         thisModule.createStateVariable(el))
52         GlobalVariables<-sc.GlobalParameters->collect(el |
53         thisModule.createGlobalVariable(el))
54         Structs<-sc.Concepts->collect(el | thisModule.createStruct(el))
55         Events<-sc.Events->collect(el | thisModule.createEvent(el))
56         Modifiers<-sc.getModifiers()
57         Functions<-sc.Functions->collect(el | thisModule.
58         createFunction(el))
59         Constructor<-thisModule.setConstructor(sc.Constructor)
60     }
61 }
62
63 lazy rule createImport {
64     from
65     i : scPIM!Import
66     to
67     im : solPSM!Import(
68         nameSpace<-i.nameSpace
69         Libraries<-i.Import->collect(el | thisModule.createLibrary(el))
70     )
71 }
72
73 lazy rule createLibrary {
74     from
75     l : scPIM!Library
76     to
77     lib : solPSM!Library(
78         libName<-l.libName
79         Contains<-l.Functions->
80         collect(el | thisModule.createFunction(el))
81     )
82 }
83
84 lazy rule createEnum {
85     from
86     en : scPIM!Enum
87     to
88     sen : solPSM!Enum(
89         enuName<-en.enuName
90         enuValue<-en.enuValue
91     )
92 }
93
94 lazy rule createGlobalVariable {
95     from
96     gp : scPIM!GlobalParameter
97     to
98     sgv : solPSM!solGlobalVariable(
99         globalName<-gp.globalName
100         gType<-thisModule.setGlobalType(gp.gType)
101     )
102 }
103
104 lazy rule createStruct {

```



```

101  from
102      co :  scPIM!Concept
103  to
104      str: solPSM!solStruct(
105          structName<-co.conceptName
106          Contains<-thisModule.setStructAttr(co.Contains)
107          Attributes<-thisModule.setStructConstruction(co.Attributes)
108          )
109  }
110
111  lazy rule createStateVariable {
112      from
113          va :  scPIM!Variable
114  to
115          ssv : solPSM!solStateVariable(
116              stateName<-varName
117              value<-value
118              dType<-thisModule.setDType(va.dType)
119              structType<-thisModule.setStructType(va.structType)
120              valueType<-thisModule.setValueType(va.valueType)
121              visibilityType<-init:  "Public"
122          )
123  }
124
125  lazy rule setConstructor {
126      from
127          cn :  scPIM!Constructor
128  to
129          sco : solPSM!solConstructor(
130              visibility<-init:  "Public"
131              Initialize<-cn.Initialize->collect(v | createStateVariable(v))
132              Construct<-cn.Construct->collect(c | createStruct(c))
133          )
134  }
135
136  lazy rule createFunction {
137      from
138          fu :  scPIM!Function
139  to
140          sfu : solPSM!solFunction(
141              functionName<-fu.funcName
142              return<-fu.return
143              interaction<-fu.interaction->setInterActionType(fu.interaction)
144              argument<-fu.argument
145              argumentType<-fu.argumentType
146              visibility<-init:  "Public"
147              modifiers<-fu.preConditions->collect(el |
148                  thisModule.createModifier(el))
149              LocalVariables<-fu.Variables->collect(el |
150                  thisModule.createLocalVariable(el))
151              LocalGlobals<-fu.LocalGlobals->collect(el |
152                  thisModule.createGlobalVariable(el))
153              LocalStructs<-fu.LocalConcepts->collect(el |
154                  thisModule.createStruct(el))
155              Contains<-fu.Contains->
156                  collect(el | thisModule.createConditionalStatement(el))

```

```

152         Execute<-fu.Execute->
            collect(e1 | thisModule.createStatement(e1))
153         Can<-fu.Can->collect(e1 | thisModule.createTransfer(e1))
154         EndedBy<-fu.EndedBy->
            collect(e1 | thisModule.createReturnStatement(e1))
155     )
156 }
157
158 lazy rule createModifier {
159     from
160         pe : scPIM!PreCondition
161     to
162         mod : solPSM!solModifier(
163             modName<-pe.preconName
164             modArgument<-pe.preconArg
165             ArgType<-pe.preconArgType
166             Conditions<-pe.condition->
                thisModule.createConditionalStatement(pe.condition)
167         )
168 }
169
170 lazy rule createLocalVariable {
171     from
172         va : scPIM!Variable
173     to
174         sv : solPSM!solLocalVariable(
175             localName<-varName
176             value<-value
177             dType<-thisModule.setDType(va.dType)
178             structType<-thisModule.setStructType(va.structType)
179             valueType<-thisModule.setValueType(va.valueType)
180             visibilityType<-init: "Public"
181         )
182 }
183
184 lazy rule createConditionalStatement {
185     from
186         cs : scPIM!ConditionalStatement
187     to
188         scs : solPSM!ConditionalStatement(
189             if<-cs.if
190             argument<-cs.argument
191             relOperator<-cs.relOperator
192             negation<-cs.negation
193             Condition<-cs.Condition->
                collect(e1 | thisModule.createConditionalStatement(e1))
194             Execute<-cs.Execute->
                collect(e1 | thisModule.createStatement(e1))
195             Can<-cs.Can->
                collect(e1 | thisModule.createTransfer(e1))
196             Message<-cs.Message->
                collect(e1 | thisModule.createMessage(e1))
197             Catch<-cs.Catch->
                collect(e1 | thisModule.createStatement(e1))
198             EndedBy<-cs.EndedBy->
                collect(e1 | thisModule.createReturnStatement(e1))
199         )

```

```

200 }
201
202 lazy rule createStatement {
203   from
204     st : scPIM!Statement
205   to
206     sst : solPSM!Statement(
207       argument<-st.argument
208       assign<-st.assign
209       operator<-st.operator
210       index<-st.index
211       addAssign<-st.addAssign
212       eveName<-st.eveName
213       eventArgument<-st.eventArgs
214     )
215 }
216
217 lazy rule createReturnStatement {
218   from
219     rs : scPIM!ReturnStatement
220   to
221     srs : solPSM!ReturnStatement(
222       value<-rs.value
223     )
224 }
225
226 lazy rule createMessage {
227   from
228     me : scPIM!Message
229   to
230     sms : solPSM!Message(
231       value<-me.value
232     )
233 }
234
235 lazy rule createTransfer {
236   from
237     tx : scPIM!Transfer
238   to
239     stx : solPSM!Transfer(
240       sender<-tx.sender
241       recipient<-tx.recipient
242       asset<-tx.asset
243       amount<-tx.amount
244     )
245 }

```

G.2 Transformation Rules From PIM to Hyperledger PSM

```

1  helper def :getResource(pe: scPIM!PreCondition) : String =
2    let rec : String = thisModule.nameSpace+"."+pe.Condition.if;
3
4
5  helper def :getRuleCondition : String =
6    self.resource+self.participant;
7

```

```

8  helper def :createParameters(ar : Sequence(String), art :
      Sequence(String)) : Sequence(hyperPSM!Parameter) =
9      let Parameters : Sequence(hyperPSM!Parameter)=
10      Map(createParameter(ar : Sequence(String), art :
      Sequence(String)))->asSequence();
11
12  helper def :createDecorators : Sequence(hyperPSM!Decorator) =
13      let allDecorators : Sequence(hyperPSM!Decorator)=
14      self.EndedBy->collect(de | createDecorator('Return',
      de.value))->asSequence()
15      ->union(self.Argument->collect(de | createDecorator('Param',
      de.paraName)));
16
17  helper def :createConditions(c : Sequence(scPIM!ConditionalStatement),
      pre : Sequence(scPIM!ConditionalStatement))
18      : Sequence(hyperPSM!ConditionalStatement)=
19      c->collect(e1 | thisModule.createConditionalStatement(e1))
      ->union(
20      pre->collect(e1 | thisModule.createConditionalStatement(e1));
21
22  helper def :setInteractionType(fu.interaction : String)=
23      switch (fu.interaction){
24      case(None) : {InteractionType.None} break;
25      case(Invokable) : {InteractionType.Invokable} break;
26      case(Transfers) : {InteractionType.Invokable} break;
27      }
28
29  helper def :createAttributes(arg: Sequence(String), typ:
      Sequence(String)) : Sequence(hyperPSM!Attribute) =
30      let Attributes : Sequence(hyperPSM!Attribute)=
31      Map(createAttribute(ar : Sequence(String), art :
      Sequence(String)))->asSequence();
32
33  helper def :setStructType(structType : String) : StructType =
34      switch (structType){
35      case("Enum"):{"Enum"} break;
36      case("Variable"):{"Variable"} break;
37      case("Array"):{"Array"} break;
38      case("KeyValue"):{"Concept"} break;
39      }
40
41  helper def :setDType(dType : String) : DataType =
42      switch (dType) {
43      case("String"):{DataType.String} break;
44      case("Integer"):{
45      case(ocllIsTypeOf(dType : "int")):{DataType.Integer}break;
46      case(ocllIsTypeOf(dType : "double")):{DataType.Double}break;
47      }break;
48      case("Byte"):{DataType."Byte"} break;
49      case(Address):{"DataType.String"} break;
50      }
51
52  helper def: createConceptAttributes(attr : Sequence(scPIM!Variable)) :
      Sequence(hyperPSM!Attribute)=
53      let attributes : Sequence(scPIM!Variable) =
54      scPIM!Variable.allInstances()->asSequence() in

```

```

55         attributes->collect(va | thisModule.createAttribute
           (va.varName: String, ty: String));

1  rule Meta2Meta {
2    from
3      me : scPIM!Meta
4    to
5      met : hyperPSM!Meta
6          target<-me.target,
7          nameSpace<-me.nameSpace,
8          acl<-thisModule.createACL(me),
9          queries<-thisModule.createQueries(me),
10         cto<-thisModule.createCTO(me),
11         smartcontracts<-me.smartcontracts->collect(sc |
           thisModule.SmartContract2SmartContract(sc))
12       )
13 }
14
15 lazy rule createACL{
16   from
17     me : scPIM!Meta
18   using
19     AllPreconditions
20     :
21     Seq(scPIM!PreCondition)
22     =
23     PreCondition.allInstancesFrom('scPIM')
24     ->select(pre | pre.cType = 'AccessControl');
25   to
26     acl : hyperPSM!ACL File
27         Contains<-AllPreconditions->collect(pre | createRule(pre))
28       )
29 }
30
31 lazy rule createQueries {
32   from
33     me : scPIM!Meta
34   using
35     sts : Seq(scPIM!Variable) =
36         Variable.allInstancesFrom('scPIM')
37         ->select(va | va.variableScope = 'Global');
38   to
39     acl : hyperPSM!Query File
40         Contains<-sts->collect(va | createQuery(va))
41       )
42 }
43
44 lazy rule createCTO {
45   from
46     me : scPIM!Meta
47   using
48     evs : Sequence(scPIM!Event) =
49         Event.allInstancesFrom('scPIM');
50     cos : Sequence(scPIM!Concept) =
51         Concept.allInstancesFrom('scPIM');
52     ens : AllEnums : Seq(scPIM!Enum) =

```

```

48         Enum.allInstancesFrom('scPIM');
49     pa : AllParticipants : Seq(scPIM!Agent) =
50         Agent.allInstancesFrom('scPIM');
51     ass : Seq(scPIM!Asset) =
52         Asset.allInstancesFrom('scPIM');
53     txs : Seq(scPIM!GlobalParameters) =
54         Function.allInstancesFrom('scPIM')
55         ->select(fu | fu.interaction = 'Invokable'or u.interaction =
            'Transfers');
56 to
57     cto : hyperPSM!CTO File
58         Enums<-ens->collect(el | thisModule.createEnum(el)),
59         Concepts<-cos->collect(el | thisModule.createConcept(el)),
60         Events<-evs->collect(el | thisModule.createEvent(el)),
61         Participants<-pa->collect(pa | thisModule.createParticipant(pa)),
62         Assets<-ass->collect(as | thisModule.createAsset(as)),
63         Transactions<-txs->collect(tx | thisModule.createTransaction(tx))
64     )
65 }
66
67 lazy rule SmartContract2SmartContract {
68     from
69         sc : scPIM!SmartContract
70     using
71         imp : sequence(scPIM!Function) =
72             sc.Import->collect(I | I.Imports)->collect(lib |
                lib.Functions)->asSequence();
73         afu : sequence(hyperPSM!Function) =
74             imp->collect(fu | thisModule.createFunction(fu))->asSequence();
75     to
76         ssc : hyperPSM!SmartContract(
77             SCName<-sc.SmartContractName,
78             abstract<-sc.abstract,
79             nameSpace<-thisModule.nameSpace,
80             States<-sc.GlobalVariables
                ->collect(el | thisModule.createState(el)),
81             Functions<-sc.Functions
                ->collect(el | thisModule.createFunction(el))->union(imp)
82         )
83 }
84
85 lazy rule createEnum{
86     from
87         en : scPIM!Enum
88     to
89         hen : hyperPSM!Enum(
90             enuName<-en.enuName,
91             enuValue<-en.enuValue
92         )
93 }
94
95 lazy rule createGlobalParameter {
96     from
97         gp : scPIM!GlobalParameter
98     to
99         hgp : hyperPSM!solGlobalVariable(
100             globalName<-gp.globalName,

```

```

101         gType<-thisModule.setGlobalType(gp.gType)
102     )
103 }
104
105 lazy rule createConcept {
106     from
107         c : scPIM!Concept
108     to
109         co : hyperPSM!Concept(
110             conceptName<-c.conceptName
111             conceptID<-thisModule.conceptID
112             Contains<-Contains->collect(el | thisModule.createConcept(el))
113             Attributes<-createConceptAttributes(co.Attributes)
114         )
115 }
116
117 lazy rule setInit {
118     from
119         cn : scPIM!Constructor
120     using
121         vrs : Sequence(scPIM!Variable) =
122             Variable.allInstancesFrom('scPIM')
123             ->select(var | var.variableScope = Global)->select(var |
124                 var.value->notEmpty());
125     to
126         sco : hyperPSM!Init(
127             Init<-vrs
128         )
129 }
130
131 lazy rule createFunction {
132     from
133         fu : scPIM!Function
134     using
135         opn : Sequence(scPIM!ConditionalStatement) =
136             fu.preConditions
137             ->select(pre | pre.cType = 'Temporal' or
138                 pre.cType = 'Evaluative' or pre.cType = 'Causal')
139             ->collect(el | el.Condition);
140     to
141         hfu : hyperPSM!solFunction(
142             funcName<-fu.funcName,
143             return<-fu.return,
144             interaction<-fu.interaction
145             ->setInteractionType(fu.interaction),
146             Argument<-createParameters(fu.argument, fu.argumenType),
147             EndedBy<-fu.EndedBy
148             ->collect(el | thisModule.createReturnStatement(el)),
149             Decorators<-createDecorators(),
150             Rules<-fu.preConditions->select(pre | pre.cType =
151                 'AccessControl')->collect(el | thisModule.createRule(el)),
152             LocalVariables<-fu.LocalVariables->collect(el |
153                 thisModule.createVariable(el)),
154             LocalGlobals<-fu.LocalGlobals->collect(el |
155                 thisModule.createGlobalParameter(el)),
156             Contains<-thisModule.createConditions(fu.Contains, opn),

```

```

148         Execute<-fu.Execute
           ->collect(el | thisModule.createStatement(el)),
149         Can<-fu.Can->collect(el | thisModule.createUpdate(el))
150     )
151 }
152
153 lazy rule createRule {
154     from
155         pe : scPIM!PreCondition
156     to
157         ru : hyperPSM!Rule(
158             ruleName<-pe.preconName,
159             description<-description,
160             participant<-pe.Condition.ConditionalStatement.Argument,
161             transaction<-pe.preconName,
162             resource<-getResource(pe)
163             condition<-getRuleCondition(),
164             action<-action,
165             operation<-operation
166         )
167 }
168
169 lazy rule createParameter (na: String, ty: String) {
170     to
171         pa : hyperPSM!Parameter(
172             paraName<-na,
173             dType<-thisModule.setDType(va.dType)
174         )
175 }
176
177 lazy rule createDecorator (ty: String, va: String) {
178     to
179         pa : hyperPSM!Decorator(
180             decField<-ty,
181             value<-va
182         )
183 }
184
185 lazy rule createVariable {
186     from
187         va : scPIM!Variable
188     to
189         sv : hyperPSM!Variable(
190             localName<-va.varName,
191             value<-va.value,
192             dType<-thisModule.setDType(va.dType),
193             structType<-thisModule.setStructType(va.structType),
194             scope<-va.VariableScope
195         )
196 }
197
198 lazy rule createState {
199     from
200         va : scPIM!Variable
201     to
202         sv : hyperPSM!Variable(
203             localName<-va.varName,

```



```

204         value<-va.varName,
205         dType<-thisModule.setDType(va.dType),
206         structType<-thisModule.setStructType(va.structType),
207         scope<-va.VariableScope
208     )
209 }
210
211 lazy rule createConditionalStatement {
212     from
213         cs : scPIM!ConditionalStatement
214     to
215         scs : hyperPSM!ConditionalStatement(
216             if<-cs.if,
217             Argument<-cs.argument,
218             relOperator<-cs.relOperator,
219             negation<-cs.negation,
220             Condition<-cs.Condition
221             ->collect(el | thisModule.createConditionalStatement(el)),
222             Execute<-cs.Execute
223             ->collect(el | thisModule.createStatement(el)),
224             Can<-cs.Can->collect(el | thisModule.createTransfer(el)),
225             Message<-cs.Message
226             ->collect(el | thisModule.createMessage(el)),
227             Catch<-cs.Catch
228             ->collect(el | thisModule.createStatement(el)),
229             EndedBy<-cs.EndedBy
230             ->collect(el | thisModule.createReturnStatement(el))
231         )
232 }
233
234 lazy rule createStatement {
235     from
236         st : scPIM!Statement
237     to
238         sst :
239         hyperPSM!Statement(
240             argument<-st.argument,
241             assign<-st.assign,
242             operator<-st.operator,
243             index<-st.index,
244             addAssign<-st.addAssign,
245             eveName<-st.eveName,
246             eventArgument<-st.eventArgs,
247             eventChanel<-thisModule.nameSpace
248         )
249 }
250
251 lazy rule createReturnStatement {
252     from
253         rs : scPIM!ReturnStatement
254     to
255         srs : hyperPSM!ReturnStatement(
256             value<-rs.value
257         )
258 }
259
260 lazy rule createMessage {

```

```

255   from
256   me : scPIM!Message
257   to
258       sms : hyperPSM!Message(
259           value<-me.value
260       )
261 }
262
263 lazy rule createQuery {
264   from
265   gp : scPIM!Variable(
266   to
267       qu : hyperPSM!Query(
268           queryName<-va.varName,
269           description<-"Query to get the value of State: "+va.varName,
270           statement<-thisModule.nameSpace+va.varName
271       )
272 }
273
274 lazy rule createUpdate {
275   from
276   tx : scPIM!Transfer
277   to
278       up : hyperPSM!Update(
279           argument<-tx.asset,
280           value<-tx.amount
281       )
282 }
283
284 createAttribute (na: String, ty: String) {
285   to
286       at : hyperPSM!Attribute(
287           attrName<-na,
288           attrType<-ty,
289           attrOptional<-init: False,
290           attrDefault<-init: False
291       )
292 }
293
294 lazy rule createAsset {
295   from
296   ax : scPIM!Asset
297   using
298       na : Sequence : String =
299           Sequence('amount', 'symbol', 'assetType', 'owners', 'initAmount')
300       ;
301       ty : Sequence : String =
302           Sequence('Integer', 'String', 'AssetType', 'String', 'Integer');
303   to
304       as : hyperPSM!Asset(
305           assetName<-assetName,
306           assetId<-assetId,
307           Attributes<-createAttributes(na, ty)
308       )
309 }
310 lazy rule createTransaction {

```

```

311  from
312      fu :  scPIM!Function
313  to
314      tx : hyperPSM!Transaction(
315          txName<-fu.funcName,
316          txId<-thisModule.txID
317      )
318  do{
319      thisModule.txID <- thisModule.txID + 1;
320  }
321 }
322
323 lazy rule createParticipant {
324  from
325      ag :  scPIM!Agent
326  using
327      na : Sequence : String =
328          Sequence('address');
329      ty: Sequence : String =
330          Sequence('String');
331  to
332      pa : hyperPSM!Participant(
333          partName<-ag.agentName,
334          parId<-ag.agentId,
335          Attributes<-createAttributes(na, ty)
336      )
337 }
338
339 lazy rule createEvent {
340  from
341      e :  scPIM!Event
342  to
343      ev : hyperPSM!Event(
344          eventName<-e.eventName,
345          argument<-e.argument,
346          argumentType<-e.argumentType
347      )
348 }

```

G.3 Constraints for the Platform Independent Smart Contract Model

```

context  Meta
  inv: Set{'Ethereum', 'Hyperledger'}->includes(target)
  inv: Agents->notEmpty()
context  Asset
  inv: Asset.allInstancesFrom('scPIM')->isUnique(assetName) and
      Asset.assetName->size()>0
  inv: Asset.allInstancesFrom('scPIM')->isUnique(assetId)
  inv: amount =>0
  inv: Asset.allInstancesFrom('scPIM')->isUnique(symbol)
  inv: owners->size() >0
  inv: Set{'Unique', 'Non-Unique'}->includes(assetType)
context  Agent
  inv: Agent.allInstancesFrom('scPIM')->isUnique(agentName) and
      Agent.agentName->size()>0
  inv: Agent.allInstancesFrom('scPIM')->isUnique(agentId)

```

```

    inv: address->notEmpty()
context Smart Contract
    inv: scPIM!SmartContract.allInstancesFrom('scPIM')->isUnique(SCName)
    inv: and SCName->size()>0
    inv: Set{'true', 'false'}->includes(abstract)
    inv: Owner->notEmpty()
    inv: Functions->notEmpty()
context Import
    inv: nameSpace->size>0
    inv: Imports->notEmpty()
context Library
    inv: scPIM!Library.allInstancesFrom('scPIM')->isUnique(libName) and
        libName->size()>0
    inv: Functions->notEmpty()
context GlobalParameter
    inv: scPIM!GlobalParameter.allInstancesFrom('scPIM')->isUnique(globalName)
        and globalName->size()>0
    inv: Set{'CurrentTime', 'SenderTransaction',
        'ValueTransaction'}->includes(gType)
context Event
    inv: Event.allInstancesFrom('scPIM')->isUnique(eventName) and
        eventName->size()>0
    inv: argument->size()>0
    inv: argumentType->size()>0
    inv: argument->size() = argumentType->size()
    inv: argumentType->forAll(at | Set{'String', 'Byte',
        'Integer', 'Address', 'ArrayString', 'ArrayByte', 'ArrayInteger',
        'Address', 'ArrayString', 'ArrayByte', 'ArrayInteger',
        'ArrayAddress', 'KeyVal:AddressToString', 'KeyVal:StringToInteger',
        'KeyVal:StringToBytes', 'KeyVal:StringToString', 'KeyVal:IntegerToString',
        'KeyVal:IntegerToInteger', 'KeyVal:IntegerToBytes', 'KeyVal:AddressToString',
        'KeyVal:AddressToBytes', 'KeyVal:AddressToInteger',
        'GlobalP:valueTransaction',
        'GlobalP:currentTime', 'GlobalP:senderTransaction'}->includes(at)
        ->includes(at) xor scPIM!Concept.allInstancesFrom'scPIM'->exists(co |
        co.conceptName = at)
        xor scPIM!Enum.allInstancesFrom'scPIM'->exists(en | en.enuName = at))
context Enum
    inv: scPIM!Enum.allInstancesFrom'scPIM'->isUnique(enuName) and
        enuName->size()>0
    inv: enuValue->size()>0
context Variable
    inv: scPIM!Variable.allInstancesFrom('scPIM')->isUnique(varName) and
        varName->size()>0
    inv: Set{'Enum', 'Variable', 'KeyValue', 'Array'}->includes(structType)
    inv: StructureType:: Variable implies self.value->size() = 1 and
        Set{'String', 'Byte', 'Integer', 'Address'}->includes(dType)
    inv: StructureType:: Array implies Set{'String', 'Byte', 'Integer',
        'Address'}->includes(dType)
        xor scPIM!Concept.allInstancesFrom'scPIM'->exists(co | co.conceptName
        = dType)
        xor scPIM!Enum.allInstancesFrom'scPIM'->exists(en | en.enuName =
        dType)
    inv: StructureType:: Enum implies
        scPIM!Enum.allInstancesFrom'scPIM'->exists(en | en.enuName = dType
        and if value->notEmpty()->value->size() = 1
        en.enuValues->includes(value))

```

```

inv:  StructureType:: KeyValue implies valueType->notEmpty() and
      Set{'String', 'Byte', 'Integer', 'Address'}->includes(dType)
      and Set{'String', 'Byte', 'Integer', 'Address'}->includes(valueType)
      xor scPIM!Enum.allInstancesFrom'scPIM'->exists(en | en.enuName =
      valueType)
      xor scPIM!Concept.allInstancesFrom'scPIM'->exists(co | co.conceptName
      = valueType)
context Concept
  inv:  scPIM!Concept.allInstancesFrom'scPIM'->isUnique(conceptName) and
        conceptName->size()>0
  inv:  Attributes->notEmpty()
context Constructor
  inv:  Set{'Private', 'Public', 'Internal'}->includes(visibilityType)
  inv:  Construct->isEmpty() implies Initialize->notEmpty()
  inv:  Initialize->isEmpty() implies Construct->notEmpty()
context PreCondition
  inv:  scPIM!PreCondition.allInstancesFrom('scPIM')->isUnique(preconName)
        and preconName->size()>0
  inv:  preconArgument ->notEmpty()
  inv:  preconArgument->size() = preconArgType->size()
  inv:  preconArgType->forAll(at | Set{'String', 'Byte', 'Integer',
        Address', 'ArrayString', 'ArrayByte', 'ArrayInteger',
        ArrayAddress', 'KeyVal:AddressToString', 'KeyVal:StringToInteger',
        KeyVal:StringToBytes', 'KeyVal:StringToString',
        'KeyVal:IntegerToString',
        KeyVal:IntegerToInteger', 'KeyVal:IntegerToBytes',
        'KeyVal:AddressToString',
        KeyVal:AddressToBytes', 'KeyVal:AddressToInteger',
        'GlobalP:valueTransaction',
        GlobalP:currentTime', 'GlobalP:senderTransaction'}->includes(at)
        xor scPIM!Concept.allInstancesFrom'scPIM'->exists(co | co.conceptName
        = at)
        xor scPIM!Enum.allInstancesFrom'scPIM'->exists(en | en.enuName = at))
  inv:  Set{'AccessControl', 'Evaluative', 'Temporal',
        'Casual'}->includes(cType)
  inv:  PreConditionType:: AccessControl implies preconArgument->size()>1
        and argumentType->includes(GlobalP:senderTransaction)
        and
        PreCondition.Condition.ConditionalStatement.relOperator->notEmpty()
        and Set{'equalTo', 'notEqualTo'}->includes(relOperator)
  inv:  operation->notEmpty() and PimModel.target = Ethereum
        implies Set{'Update', 'Read'}->includes(operation)
  inv:  operation->notEmpty() and PimModel.target = Hyperledger
        implies Set{'Read', 'Delete', 'Update', 'Create'}->includes(operation)
  inv:  PreConditionType:: Evaluative implies preconArgument->size()>1
        and Condition.ConditionalStatement.relOperator->notEmpty()
  inv:  PreConditionType:: Temporal implies
        scPIM!GlobalParameter.allInstancesFrom'scPIM'->exists(gp |
        gp.globalName = Condition.ConditionalStatement.if)
        and gp.gType = 'currentTime' and relOperator->notEmpty()
  inv:  PreConditionType:: Causal implies
        scPIM!Variable.allInstancesFrom'scPIM'->exists(v | v.varName =
        condition.ConditionalStatement.if and v.dType = 'Enum')
        and Condition.ConditionalStatement.relOperator->notEmpty()
context Function
  inv:  scPIM!Function.allInstancesFrom('scPIM')->isUnique(funcName) and
        funcName->size()>0

```

```

inv: Set{'None', 'Invokable', 'Transfers'}->includes(interaction)
inv: argument->size() = argumentType->size()
inv: argumentType->forAll(at | Set{'String', 'Byte', 'Integer',
    Address', 'ArrayString', 'ArrayByte', 'ArrayInteger',
    ArrayAddress', 'KeyVal:AddressToString', 'KeyVal:StringToInteger',
    KeyVal:StringToBytes', 'KeyVal:StringToString',
    'KeyVal:IntegerToString',
    KeyVal:IntegerToInteger', 'KeyVal:IntegerToBytes',
    'KeyVal:AddressToString',
    KeyVal:AddressToBytes', 'KeyVal:AddressToInteger',
    'GlobalP:valueTransaction',
    GlobalP:currentTime', 'GlobalP:senderTransaction'})->includes(at)
xor scPIM!Concept.allInstancesFrom('scPIM')->exists(co | co.conceptName
    = at)
xor scPIM!Enum.allInstancesFrom('scPIM')->exists(en | en.enuName = at))
inv: return = true implies self.EndedBy->count(ReturnStatement)>0
inv: Contains->isEmpty() implies Execute->notEmpty()
inv: Execute->isEmpty() implies Contains->notEmpty()
inv: LocalVariables->forAll(va | va.variableScope = VarScope.Local)
context Conditional Statement
inv: scPIM!Variable.allInstancesFrom('scPIM')->exists(v | v.varName = if)
xor scPIM!GlobalParameter.allInstancesFrom('scPIM')->exists(gp |
    gp.globalName = if)
xor scPIM!Agent.allInstancesFrom('scPIM')->exists(ag | ag.agentId =
    if)
inv: scPIM!Variable.allInstancesFrom('scPIM')->exists(v | v.varName =
    argument)
xor scPIM!GlobalParameter.allInstancesFrom('scPIM')->exists(gp |
    gp.globalName = argument)
xor scPIM!Agent.allInstancesFrom('scPIM')->exists(ag | ag.agentId =
    argument)
Set{'greaterThan', 'equalTo', 'smallerThan',
    'notEqualTo'}->includes(relOperator)
Set{'True', 'False'}->includes(negation)
inv: Catch->notEmpty() implies Message->isEmpty()
inv: Message->notEmpty() implies Catch->isEmpty()
context Statement
inv: argument->size()>0 and argument->size()<3
    argument->forAll(arg |
        scPIM!Variable.allInstancesFrom('scPIM')->exists(v | v.varName = arg)
        xor scPIM!GlobalParameters.allInstancesFrom('scPIM')->exists(gp |
            gp.globalName = arg)
        xor Function.allInstancesFrom('scPIM')->exists(fu | fu.funcName =
            arg)
        xor Function.allInstancesFrom('scPIM')->exists(fu |
            fu.argument.includes(arg))
inv: assign->scPIM!Variable.allInstancesFrom('scPIM')->exists(var |
    var.varName = assign)
xor scPIM!GlobalParameters.allInstancesFrom('scPIM')->exists(gp |
    gp.globalName = assign)
xor Function.allInstancesFrom('scPIM')->exists(fu | fu.funcName =
    assign)
xor Function.allInstancesFrom('scPIM')->exists(fu |
    fu.argument.includes(assign))
operator->notEmpty() implies argument->size() = 2
and Set{'Addition', 'Subtraction', 'Modulus', 'Increment',
    'Decrement', 'Division', 'Multiplication'}

```

```

->includes(operator)
inv:  addAssign->scPIM!Variable.allInstancesFrom'scPIM'->exists(v |
      v.varName = addAssign)
      xor scPIM!GlobalParameters.allInstancesFrom'scPIM'->exists(gp |
      gp.globalName = addAssign)
inv:  eveName->notEmpty() implies
      scPIM!Event.allInstancesFrom'scPIM'->exists(ev| ev.eventName =
      eveName)
      and eveArgs->notEmpty()
context Transfer
inv:  scPIM!Agent.allInstancesFrom'scPIM'->exists(ag | ag.agentId = sender)
inv:  scPIM!Agent.allInstancesFrom'scPIM'->exists(ag | ag.agentId =
      recipient)
inv:  scPIM!Asset.allInstancesFrom'scPIM'->exists(as | as.assetId = asset)
inv:  sender <>recipient
inv:  amount =>0
context Return Statement
inv:  value->size()>0
inv:  value->forAll(val
      |scPIM!StateVariable.allInstancesFrom'scPIM'->exists(svar |
      svar.varName = val)
      xor val.oclIsTypeOf(String) xor val.oclIsTypeOf(Integer))

```

Bibliography

- [1] *A Scalable Architecture for On-Demand, Untrusted Delivery of Entropy*. Nov. 2018. url: <https://docs.oracle.com/iaas/entropy/#background>.
- [2] Yassine Ait Hsain, Naziha Laaz, and Samir Mbarki. "Ethereum's Smart Contracts Construction and Development using Model Driven Engineering Technologies: a Review". In: *Procedia Computer Science* 184 (2021). The 12th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 4th International Conference on Emerging Data and Industry 4.0 (EDI40) / Affiliated Workshops, pp. 785–790. issn: 1877-0509. doi: <https://doi.org/10.1016/j.procs.2021.03.097>. url: <https://www.sciencedirect.com/science/article/pii/S1877050921007389>.
- [3] Firas Al Khalil et al. "Trust in smart contracts is a process, as well". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2017, pp. 510–519.
- [4] Robert Alexy. *A theory of constitutional rights*. Oxford University Press, USA, 2010.
- [5] Maher Alharby and Aad van Moorsel. "A systematic mapping study on current research topics in smart contracts". In: *International Journal of Computer Science & Information Technology* 9.5 (2017), pp. 151–164.
- [6] James F. Allen. "Maintaining knowledge about temporal intervals". In: *Communications of the ACM* 26.11 (1983), pp. 832–843.
- [7] Elli Androulaki et al. "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains". In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. Porto, Portugal: ACM, 2018, 30:1–30:15. isbn: 978-1-4503-5584-1. doi: 10.1145/3190508.3190538. url: <http://doi.acm.org/10.1145/3190508.3190538>.
- [8] Stephanos Androulakis-Theotokis and Diomidis Spinellis. "A Survey of Peer-to-peer Content Distribution Technologies". In: *ACM Comput. Surv.* 36.4 (Dec. 2004), pp. 335–371. issn: 0360-0300. doi: 10.1145/1041680.1041681. url: <http://doi.acm.org/10.1145/1041680.1041681>.
- [9] Monika di Angelo, Alfred Soare, and Gernot Salzer. "Smart Contracts in View of the Civil Code". In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. SAC '19. Limassol, Cyprus: Association for Computing Machinery, 2019, 392–399. isbn: 9781450359337. doi: 10.1145/3297280.3297321. url: <https://doi.org/10.1145/3297280.3297321>.
- [10] Samuil Angelov and Paul Grefen. "An e-contracting reference architecture". In: *Journal of Systems and Software* 81.11 (2008), pp. 1816–1844.
- [11] Matthias Armgardt and Giovanni Sartor. "Leibniz, Gottfried Wilhelm von: Legal Philosophy". In: *Encyclopedia of the Philosophy of Law and Social Philosophy*. Ed. by Mortimer Sellers and Stephan Kirste. Dordrecht: Springer Netherlands, 2019, pp. 1–5. isbn: 978-94-007-6730-0. doi: 10.1007/978-94-007-6730-0_434-1. url: https://doi.org/10.1007/978-94-007-6730-0_434-1.
- [12] Clarence Ashley. "Conditions in Contract". In: *Yale Law Journal* 14 (1904), p. 424.
- [13] IEEE Standards Association et al. *Systems and software engineering-Vocabulary*. Tech. rep. ISO/IEC/IEEE 24765: 2010. Iso/iec/ieee, 24765, 1-418, 2010.
- [14] Tomaso Aste, Paolo Tasca, and Tiziana Di Matteo. "Blockchain technologies: The foreseeable impact on society and industry". In: *Computer* 50.9 (2017), pp. 18–28.

- [15] Colin Atkinson and Thomas Kühne. "Rearchitecting the UML infrastructure". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 12.4 (2002), pp. 290–321.
- [16] Various authors. *Whitepaper: Smart Contracts and Distributed Ledger - A Legal Perspective*. Tech. rep. New York, NY, USA, 2017.
- [17] Sarah Azouvi, Mustafa Al-Bassam, and Sarah Meiklejohn. "Who am I? Secure identity registration on distributed ledgers". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Cham: Springer, 2017, pp. 373–389.
- [18] Adam Back et al. "Enabling blockchain innovations with pegged sidechains". In: (2014). eprint: <https://www.semanticscholar.org/paper/Enabling-Blockchain-Innovations-with-Pegged-Back-Corall>.
- [19] James Bailey et al. "An Event-Condition-Action Language for XML". In: *Web Dynamics: Adapting to Change in Content, Size, Topology and Use*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 223–248. isbn: 978-3-662-10874-1. doi: 10.1007/978-3-662-10874-1_10. url: https://doi.org/10.1007/978-3-662-10874-1_10.
- [20] Peter Bailis et al. "Research for practice: cryptocurrencies, blockchains, and smart contracts; hardware for deep learning". In: *Communications of the ACM* 60.5 (2017), pp. 48–51.
- [21] Foteini Baldimtsi et al. "Indistinguishable Proofs of Work or Knowledge". In: *Advances in Cryptology – ASIACRYPT 2016*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 902–933. isbn: 978-3-662-53890-6.
- [22] Massimo Bartoletti and Livio Pompianu. "An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns". In: *Financial Cryptography and Data Security*. Ed. by Michael Brenner et al. Cham: Springer International Publishing, 2017, pp. 494–509. isbn: 978-3-319-70278-0.
- [23] Richard Baskerville. "What design science is not". In: *European Journal of Information Systems* 17.5 (2008), pp. 441–443. doi: 10.1057/ejis.2008.45. eprint: <https://doi.org/10.1057/ejis.2008.45>. url: <https://doi.org/10.1057/ejis.2008.45>.
- [24] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. SEI series in software engineering. Addison-Wesley, 2003. isbn: 9780321154958. url: <http://books.google.fi/books?id=mdiIu8Kk1WMC>.
- [25] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Boston, MA, USA: Pearson Education, 2003. isbn: 978-0-321-15495-8.
- [26] F. Rizal Batubara, Jolien Ubacht, and Marijn Janssen. "Challenges of Blockchain Technology Adoption for e-Government: A Systematic Literature Review". In: *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age*. dg.o '18. Delft, The Netherlands: ACM, 2018, 76:1–76:9. isbn: 978-1-4503-6526-0. doi: 10.1145/3209281.3209317. url: <http://doi.acm.org/10.1145/3209281.3209317>.
- [27] Mariano Belaunde et al. *MDA Guide Version 1.0.1*. Tech. rep. Object Management Group, Inc., 2003.
- [28] Rafael Belchior et al. "A Survey on Blockchain Interoperability: Past, Present, and Future Trends". In: *ACM Comput. Surv.* 54.8 (Sept. 2021). issn: 0360-0300. doi: 10.1145/3471140. url: <https://doi.org/10.1145/3471140>.
- [29] Trevor Bench-Capon and Thomas F. Gordon. "Isomorphism and Argumentation". In: *Proceedings of the 12th International Conference on Artificial Intelligence and Law*. ICAIL '09. Barcelona, Spain: Association for Computing Machinery, 2009, 11–20. isbn: 9781605585970. doi: 10.1145/1568234.1568237. url: <https://doi.org/10.1145/1568234.1568237>.

- [30] Iddo Bentov et al. "proof of activity: Extending bitcoin's proof of work via proof of stake". In: *ACM SIGMETRICS Performance Evaluation Review* 42.3 (2014), pp. 34–37.
- [31] Ghassan Beydoun et al. "FAML: a generic metamodel for MAS development". In: *IEEE Transactions on Software Engineering* 35.6 (2009), pp. 841–863.
- [32] Gavin Bierman, Martin Abadi, and Mads Torgersen. "Understanding TypeScript". In: *ECOOP 2014 – Object-Oriented Programming*. Ed. by Richard Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 257–281. isbn: 978-3-662-44202-9.
- [33] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. "Deanonymisation of Clients in Bitcoin P2P Network". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. Scottsdale, Arizona, USA: ACM, 2014, pp. 15–29. isbn: 978-1-4503-2957-6. doi: 10.1145/2660267.2660379. url: <http://doi.acm.org/10.1145/2660267.2660379>.
- [34] Guido Boella et al. "A critical analysis of legal requirements engineering from the perspective of legal practice". In: *2014 IEEE 7th International Workshop on Requirements Engineering and Law (RELAW)*. IEEE, 2014, pp. 14–21.
- [35] Juan Boubeta-Puig, Jesús Rosa-Bilbao, and Jan Mendling. "CEPchain: A graphical model-driven solution for integrating complex event processing and blockchain". In: *Expert Systems with Applications* 184 (2021), p. 115578. issn: 0957-4174. doi: <https://doi.org/10.1016/j.eswa.2021.115578>. url: <https://www.sciencedirect.com/science/article/pii/S0957417421009805>.
- [36] Phillip Boucher. *How blockchain technology could change our lives*. Tech. rep. Brussels, Belgium, 2017.
- [37] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. "Model-driven software engineering in practice". In: *Synthesis lectures on software engineering* 3.1 (2017), pp. 1–207.
- [38] Marcus Brandenburger et al. "Rollback and forking detection for trusted execution environments using lightweight collective memory". In: *Proceedings of the 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* 2017. Denver: IEEE, 2017, pp. 157–168.
- [39] Richard Braun et al. "Proposal for Requirements Driven Design Science Research". In: *New Horizons in Design Science: Broadening the Research Agenda*. Ed. by Brian Donnellan et al. Cham: Springer International Publishing, 2015, pp. 135–151. isbn: 978-3-319-18714-3.
- [40] Travis Breaux and Annie Antón. "Analyzing regulatory rules for privacy and security requirements". In: *IEEE transactions on software engineering* 34.1 (2008), pp. 5–20.
- [41] Frederick P Brooks Jr. "The computer scientist as toolsmith II". In: *Communications of the ACM* 39.3 (1996), pp. 61–68.
- [42] Richard Gendal Brown et al. *Corda: A Distributed Ledger*. Tech. rep. New York, NY, USA: R3, 2016.
- [43] Antonio Bucchiarone et al. "Grand challenges in model-driven engineering: an analysis of the state of the research". In: *Software and Systems Modeling* 19.1 (2020), pp. 5–13.
- [44] Christian Cachin and Marko Vukolić. "Blockchains consensus protocols in the wild". In: (2017). eprint: [arXivpreprintarXiv:1707.01873](https://arxiv.org/abs/1707.01873).
- [45] W. Cai et al. "Decentralized Applications: The Blockchain-Empowered Software System". In: *IEEE Access* 6 (2018), pp. 53019–53033.
- [46] Davide Calvaresi et al. "Multi-agent systems and blockchain: Results from a systematic literature review". In: *International Conference on Practical Applications of Agents and Multi-Agent Systems*. Cham: Springer, 2018, pp. 110–126.
- [47] Fran Casino, Thomas K. Dasaklis, and Constantinos Patsakis. "A systematic literature review of blockchain-based applications: Current status, classification and open

- issues". In: *Telematics and Informatics* 36 (2019), pp. 55–81. issn: 0736-5853. doi: <https://doi.org/10.1016/j.tele.2018.11.006>. url: <http://www.sciencedirect.com/science/article/pii/S0736585318306324>.
- [48] Rishav Chatterjee and Rajdeep Chatterjee. "An Overview of the Emerging Technology: Blockchain". In: *Computational Intelligence and Networks (CINE), 2017 3rd International Conference on*. IEEE, 2017, pp. 126–127.
- [49] Lin Chen et al. "On Security Analysis of Proof-of-Elapsed-Time (PoET)". In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by Paul Spirakis and Philippas Tsigas. Cham: Springer International Publishing, 2017, pp. 282–297. isbn: 978-3-319-69084-1.
- [50] D. K. W. Chiu, S. C. Cheung, and S. Till. "A three-layer architecture for e-contract enforcement in an e-service environment". In: *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*. Vol. 9. 2003, p. 10. doi: 10.1109/HICSS.2003.1174188.
- [51] Dickson KW Chiu, Shing-Chi Cheung, and Sven Till. "A three-layer architecture for e-contract enforcement in an e-service environment". In: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences 2003*. IEEE, 2003, 10–pp.
- [52] K.-Y. Chow, A.J. David, and A.M. Ionescu-Graff. "Switching Capacity Relief Model: Theoretical Development". In: 1983, pp. 1–7. url: http://i-teletraffic.org/_Resources/Persistent/d1af3fa05b6d58cea28b80af39fd919a290d8b4a/chow831.pdf.
- [53] Konstantinos Christidis and Michael Devetsikiotis. "Blockchains and smart contracts for the internet of things". In: *IEEE Access* 4 (2016), pp. 2292–2303.
- [54] Sung-Chi Chu et al. "Evolution of e-commerce Web sites: A conceptual framework and a longitudinal study". In: *Information & Management* 44.2 (2007), pp. 154–164.
- [55] G. Ciatto, S. Mariani, and A. Omicini. "Blockchain for Trustworthy Coordination: A First Study with LINDA and Ethereum". In: *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. Dec. 2018, pp. 696–703. doi: 10.1109/WI.2018.000–9.
- [56] Christopher D Clack, Vikram A Bakshi, and Lee Braine. "Smart Contract Templates: essential requirements and design options". In: (2016). eprint: 1612.04496.
- [57] Christopher D Clack, Vikram A Bakshi, and Lee Braine. "Smart Contract Templates: essential requirements and design options". In: (2016). eprint: 1612.04496.
- [58] Christopher D Clack, Vikram A Bakshi, and Lee Braine. "Smart contract templates: foundations, design landscape and research directions". In: (2016). eprint: 1608.00771.
- [59] Michael Coblenz et al. "Smarter Smart Contract Development Tools". In: *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. WETSEB. IEEE. May 2019, pp. 48–51. doi: 10.1109/WETSEB.2019.00013.
- [60] Bram Cohen and Krzysztof Pietrzak. "Simple proofs of sequential work". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Cham: Springer, 2018, pp. 451–467.
- [61] The Ethereum Community. "Ethereum Homestead Documentation, Release 01". In: (2017). eprint: <http://www.ethdocs.org/en/latest/>.
- [62] Michael Crosby et al. *Blockchain Technology Beyond Bitcoin*. Tech. rep. Berkeley, CA, USA, 2015.
- [63] Jason Paul Cruz, Yuichi Kaji, and Naoto Yanai. "RBAC-SC: Role-Based Access Control Using Smart Contract". In: *IEEE Access* 6 (2018), pp. 12240–12251.
- [64] Pierluigi Cuccuru. "Beyond bitcoin: an early overview on smart contracts". In: *International Journal of Law and Information Technology* 25.3 (2017), pp. 179–195.

- [65] Aspasia Daskalopulu and Marek Sergot. "A Constraint-Driven System for Contract Assembly". In: *Proceedings of the 5th International Conference on Artificial Intelligence and Law*. ICAIL '95. College Park, Maryland, USA: Association for Computing Machinery, 1995, 62–70. isbn: 0897917588. doi: 10 . 1145 / 222092 . 222130. url: <https://doi.org/10.1145/222092.222130>.
- [66] Primavera De Filippi and Aaron Wright. *Blockchain and the law: The rule of code*. Harvard University Press, 2018. isbn: 978-0-674-98591-9.
- [67] Joost De Kruijff and Hans Weigand. "Understanding the blockchain using enterprise ontology". In: *International Conference on Advanced Information Systems Engineering*. Springer. 2017, pp. 29–43.
- [68] Christian Decker et al. "Making Bitcoin Exchanges Transparent". In: *European Symposium on Research in Computer Security*. Ed. by Günther Pernul, Peter Y A Ryan, and Edgar Weippl. ESORICS 2015. Cham: Springer International Publishing, 2015, pp. 561–576. isbn: 978-3-319-24177-7.
- [69] Andre L Delbecq, Andrew H Van de Ven, and David H Gustafson. *Group techniques for program planning: A guide to nominal group and Delphi processes*. Green Briar Press, 1975.
- [70] Thomas Dickerson et al. "Adding concurrency to smart contracts". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. New York, NY, USA: ACM, 2017, pp. 303–312.
- [71] Tien Tuan Anh Dinh et al. "Blockbench: A framework for analyzing private blockchains". In: *Proceedings of the 2017 ACM International Conference on Management of Data*. New York, NY, USA: ACM, 2017, pp. 1085–1100.
- [72] Tien Tuan Anh Dinh et al. "Untangling blockchain: A data processing view of blockchain systems". In: *IEEE Transactions on Knowledge and Data Engineering* 30.7 (2018), pp. 1366–1385.
- [73] Stefan Driessen et al. *Automated Test-Case Generation for Solidity Smart Contracts: the AGSoIT Approach and its Evaluation*. 2021. doi: 10.48550/ARXIV.2102.08864. url: <https://arxiv.org/abs/2102.08864>.
- [74] Daniel Drummer and Dirk Neumann. "Is code law? Current legal and technical adoption issues and remedies for blockchain-enabled smart contracts". In: *Journal of Information Technology* 35.4 (2020), pp. 337–360. doi: 10.1177/0268396220924669. eprint: <https://doi.org/10.1177/0268396220924669>. url: <https://doi.org/10.1177/0268396220924669>.
- [75] Vimal Dwivedi et al. "Legally Enforceable Smart-Contract Languages: A Systematic Literature Review". In: *ACM Computing Surveys (CSUR)* 54.5 (2021), pp. 1–34.
- [76] Jordan Earls, Mark Smith, and Ronald Smith. *Smart Contracts: Is the law ready?* Tech. rep. London, United Kingdom, 2017.
- [77] Khaled El Emam and A Günes Koru. "A replicated survey of IT software project failures". In: *IEEE software* 25.5 (2008), pp. 84–90.
- [78] *Embracing Disruption, Tapping the Potential of Distributed Ledger to Improve the Post-Trade Landscape*. Tech. rep. New York, NY, USA, 2016.
- [79] E. Allan Farnsworth. "Legal Remedies for Breach of Contract". In: *Columbia Law Review* 70.7 (1970), pp. 1145–1216. issn: 00101958. url: <http://www.jstor.org/stable/1121184>.
- [80] Scott Farrell, Heidi Machin, and Roslyn Hinchliffe. "Lost and found in smart contract translation — considerations in transitioning to automation in legal architecture". In: *Proceedings of the congress of the United Nations commission on international trade law*. Vol. 4. UNCITRAL. United Nations, 2017, pp. 95–104.

- [81] Luca Foschini et al. "Hyperledger Fabric Blockchain: Chaincode Performance Analysis". In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*. 2020, pp. 1–6. doi: 10.1109/ICC40277.2020.9149080.
- [82] Christopher K Frantz and Mariusz Nowostawski. "From institutions to code: Towards automated generation of smart contracts". In: *2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. 2016, pp. 210–215. doi: 10.1109/FAS-W.2016.53.
- [83] Michael Fröwis and Rainer Böhme. "In Code We Trust?" In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2017, pp. 357–372.
- [84] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. "The bitcoin backbone protocol: Analysis and applications". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Cham: Springer, 2015, pp. 281–310.
- [85] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. "The bitcoin backbone protocol with chains of variable difficulty". In: *Annual International Cryptology Conference*. Cham: Springer, 2017, pp. 291–323.
- [86] Vahid Garousi, Michael Felderer, and Mika V. Mäntylä. "Guidelines for including grey literature and conducting multivocal literature reviews in software engineering". In: *Information and Software Technology* 106 (2019), pp. 101–121. issn: 0950-5849. doi: <https://doi.org/10.1016/j.infsof.2018.09.006>. url: <http://www.sciencedirect.com/science/article/pii/S0950584918301939>.
- [87] Dragan Gašević, Dragan Djuric, and Vladan Devedžic. *Model driven architecture and ontology development*. Springer Science & Business Media, 2006.
- [88] Valentina Gatteschi et al. "To Blockchain or Not to Blockchain: That Is the Question". In: *IT Professional* 20.2 (2018), pp. 62–74.
- [89] Arthur Gervais et al. "On the security and performance of proof of work blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2016, pp. 3–16.
- [90] Mark Giancaspro. "Is a 'smart contract' really a smart idea? Insights from a legal perspective". In: *Computer Law & Security Review* 33.6 (2017), pp. 825–835. issn: 0267-3649. doi: <https://doi.org/10.1016/j.clsr.2017.05.007>. url: <http://www.sciencedirect.com/science/article/pii/S026736491730167X>.
- [91] Georgios K. Giannikis and Aspasia Daskalopulu. "Normative conflicts in electronic contracts". In: *Electronic Commerce Research and Applications* 10.2 (2011). Special Issue on Electronic Auctions: Strategies and Methods, pp. 247–267. issn: 1567-4223. doi: <https://doi.org/10.1016/j.elerap.2010.09.005>. url: <https://www.sciencedirect.com/science/article/pii/S1567422310000815>.
- [92] J. Gilcrest and A. Carvalho. "Smart Contracts: Legal Considerations". In: *2018 IEEE International Conference on Big Data (Big Data)*. 2018, pp. 3277–3281.
- [93] Barney G Glaser and Anselm L Strauss. *Discovery of Grounded Theory: Strategies for Qualitative Research*. New York, NY, USA: Routledge, 2017.
- [94] Florian Glaser. "Pervasive decentralisation of digital infrastructures: a framework for blockchain enabled system and use case analysis". In: *Proceedings of the 50th Hawaii International Conference on System Sciences*. 2017, pp. 1543–1552.
- [95] Tony Gorschek and Claes Wohlin. "Requirements abstraction model". In: *Requirements Engineering* 11.1 (2006), pp. 79–101.
- [96] Guido Governatori et al. "On legal contracts, imperative and declarative smart contracts, and blockchain systems". In: *Artificial Intelligence and Law* 26.4 (2018), pp. 377–409.
- [97] Gideon Greenspan. "MultiChain Private Blockchain - White Paper". In: (2015). eprint: <https://www.multichain.com/download/MultiChain-White-Paper.pdf>.

- [98] Cristine Griffo et al. "From an ontology of service contracts to contract modeling in enterprise architecture". In: *2017 IEEE 21st International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE. 2017, pp. 40–49.
- [99] Ian Grigg. "The ricardian contract". In: *Proceedings. First IEEE International Workshop on Electronic Contracting, 2004*. IEEE. 2004, pp. 25–31.
- [100] World Bank Group. *Smart Contract Technology and Financial Inclusion*. Tech. rep. Washington, DC, USA, 2020.
- [101] Seda Gürses, Carmela Troncoso, and Claudia Diaz. "Engineering Privacy by Design". In: *Computers, Privacy & Data Protection* (2011).
- [102] T. Górski and J. Bednarski. "Applying Model-Driven Engineering to Distributed Ledger Deployment". In: *IEEE Access* 8 (2020), pp. 118245–118261. doi: 10.1109/ACCESS.2020.3005519.
- [103] Brent Hailpern and Peri Tarr. "Model-driven development: The good, the bad, and the ugly". In: *IBM systems journal* 45.3 (2006), pp. 451–461.
- [104] Slimane Hammoudi, Jérôme Janvier, and Denivaldo Lopes. "Mapping versus transformation in mda: Generating transformation definition from mapping specification". In: *Foundational Ontologies* 33 (2005).
- [105] Brain Harley. *Are Smart Contracts Contracts?* Tech. rep. London, United Kingdom: Clifford Chance, Aug. 2017.
- [106] Brain Harley. *Are Smart Contracts Contracts?* Tech. rep. London, United Kingdom, 2017.
- [107] H. L. A. Hart. "Positivism and the Separation of Law and Morals". In: *Harvard Law Review* 71.4 (1958), pp. 593–629. issn: 0017811X. url: <http://www.jstor.org/stable/1338225>.
- [108] Andrew F. Hayes and Klaus Krippendorff. "Answering the Call for a Standard Reliability Measure for Coding Data". In: *Communication Methods and Measures* 1.1 (2007), pp. 77–89. doi: 10.1080/19312450709336664. eprint: <https://doi.org/10.1080/19312450709336664>. url: <https://doi.org/10.1080/19312450709336664>.
- [109] Shuangyu He et al. "A Social-Network-Based Cryptocurrency Wallet-Management Scheme". In: *IEEE Access* 6 (2018), pp. 7654–7663.
- [110] Xiao He et al. "SPESC: A Specification Language for Smart Contracts". In: *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*. IEEE. 2018, pp. 132–137.
- [111] Brian Henderson-Sellers. "Bridging metamodels and ontologies in software engineering". In: *Journal of Systems and Software* 84.2 (2011), pp. 301–313.
- [112] Garick Hileman and Michel Rauchs. *Global Blockchain Benchmark Study*. Tech. rep. Cambridge, United Kingdom, 2017.
- [113] Wesley Newcomb Hohfeld. "Some fundamental legal conceptions as applied in judicial reasoning". In: *Yale Law Journal* 23 (1913), p. 16.
- [114] Clyde W. Holsapple and K. D. Joshi. "A Collaborative Approach to Ontology Design". In: *Communications of the ACM* 45.2 (Feb. 2002), 42–47. issn: 0001-0782. doi: 10.1145/503124.503147. url: <https://doi.org/10.1145/503124.503147>.
- [115] W Hon Kuan, John Palfreyman, and Matthew Tegart. *Distributed Ledger Technology and Cybersecurity: Improving information security in the financial sector*. Tech. rep. Heraklion, Greece, 2016.
- [116] Hsiao-Shan Huang, Tian-Sheuan Chang, and Jhih-Yi Wu. "A Secure File Sharing System Based on IPFS and Blockchain". In: *Proceedings of the 2020 2nd International Electronics Communication Conference. IECC 2020*. Singapore, Singapore: Association for Computing Machinery, 2020, 96–100. isbn: 9781450377706. doi: 10.1145/3409934.3409948. url: <https://doi.org/10.1145/3409934.3409948>.

- [117] Richard Hull et al. "Towards a Shared Ledger Business Collaboration Language Based on Data-Aware Processes". In: *Service-Oriented Computing*. Ed. by Quan Z. Sheng et al. Cham: Springer International Publishing, 2016, pp. 18–36.
- [118] John Hutchinson et al. "Empirical Assessment of MDE in Industry". In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE '11. Waikiki, HI, USA: Association for Computing Machinery, 2011, 471–480. isbn: 9781450304450. doi: 10.1145/1985793.1985858. url: <https://doi.org/10.1145/1985793.1985858>.
- [119] *Hyperledger Fabric Documentation*. Hyperledger Fabric. Feb. 2020. url: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/#>.
- [120] Frédéric Jouault and Ivan Kurtev. "Transforming Models with ATL". In: *Satellite Events at the MoDELS 2005 Conference*. Ed. by Jean-Michel Bruel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 128–138. isbn: 978-3-540-31781-4.
- [121] Aljosha Judmayer et al. "Merged Mining: Curse or Cure?" In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Ed. by Joaquin Garcia-Alfaro et al. Cham: Springer International Publishing, 2017, pp. 316–333. isbn: 978-3-319-67816-0.
- [122] Mantas Jurgelaitis et al. "Smart Contract Code Generation from Platform Specific Model for Hyperledger Go". In: *World Conference on Information Systems and Technologies*. Springer. 2021, pp. 63–73.
- [123] L. Kagal and T. Finin and. "A policy language for a pervasive computing environment". In: *Proceedings POLICY 2003. IEEE 4th International Workshop on Policies for Distributed Systems and Networks*. June 2003, pp. 63–74. doi: 10.1109/POLICY.2003.1206958.
- [124] Alexey Kalinov. "Scalability of Heterogeneous Parallel Systems". In: *Programming and Computer Software* 32.1 (Jan. 2006), 1–7. doi: 10.1134/S0361768806010014. url: <https://doi.org/10.1134/S0361768806010014>.
- [125] Erik Kamsties et al. "Detecting ambiguities in requirements documents using inspections". In: *Proceedings of the first workshop on inspection in software engineering (WISE'01)*. 2001, pp. 68–80.
- [126] Elena Karafiloski and Anastas Mishev. "Blockchain solutions for big data challenges: A literature review". In: *IEEE EUROCON 2017-17th International Conference on Smart Technologies*. IEEE. 2017, pp. 763–768.
- [127] Kamalakara Karlapalem, Ajay R. Dani, and P. Radha Krishna. "A Frame Work for Modeling Electronic Contracts". In: *Conceptual Modeling — ER 2001*. Ed. by Hideko S.Kunii, Sushil Jajodia, and Arne Sølvberg. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 193–207. isbn: 978-3-540-45581-3.
- [128] Rami Khalil and Arthur Gervais. "Revive: Rebalancing off-blockchain payment networks". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2017, pp. 439–453.
- [129] Minhaj Ahmad Khan and Khaled Salah. "IoT security: Review, blockchain solutions, and open challenges". In: *Future Generation Computer Systems* 82 (2018), pp. 395–411.
- [130] Aggelos Kiayias et al. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol". In: *Advances in Cryptology – CRYPTO 2017*. Ed. by Jonathan Katz and Hovav Shacham. Cham: Springer International Publishing, 2017, pp. 357–388. isbn: 978-3-319-63688-7.
- [131] Barbara Kitchenham. "Procedures for performing systematic reviews". In: *Keele, UK, Keele University* 33.2004 (2004), pp. 1–26.
- [132] Nadzeya Kiyavitskaya et al. "Automating the Extraction of Rights and Obligations for Regulatory Compliance". In: *Conceptual Modeling - ER 2008*. Ed. by Qing Li et al.

- Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 154–168. isbn: 978-3-540-87877-3.
- [133] Anneke G Kleppe et al. *MDA explained: the model driven architecture: practice and promise*. Addison-Wesley Professional, 2003.
- [134] Kari Korpela, Jukka Hallikas, and Tomi Dahlberg. "Digital supply chain transformation toward blockchain integration". In: *Proceedings of the 50th Hawaii international conference on system sciences*. Honolulu: IEEE, 2017, pp. 4182–4191. isbn: 978-0-9981331-0-2. doi: 10.24251/HICSS.2017.506.
- [135] Ahmed Kosba et al. "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts". In: *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016, pp. 839–858.
- [136] Robert van Kralingen. "A conceptual frame-based ontology for the law". In: *Proceedings of the first international workshop on legal ontologies*. Citeseer. 1997, pp. 6–17.
- [137] Klaus Krippendorff. *Content analysis: An introduction to Its Methodology*. 3rd ed. London, United Kingdom: Sage Publications, 2018. isbn: 9781506395661.
- [138] P Radha Krishna and Kamalakar Karlapalem. "A methodology for evolving e-contracts using templates". In: *IEEE Transactions on Services Computing* 6.4 (2012), pp. 497–510.
- [139] P Radha Krishna, Kamalakar Karlapalem, and Dickson KW Chiu. "An EREC framework for e-contract modeling, enactment and monitoring". In: *Data & Knowledge Engineering* 51.1 (2004), pp. 31–58.
- [140] P Radha Krishna, Kamalakar Karlapalem, and Ajay R Dani. "From contracts to e-contracts: Modeling and enactment". In: *Information Technology and Management* 6.4 (2005), pp. 363–387.
- [141] Philippe B Kruchten. "The 4+ 1 view model of architecture". In: *IEEE software* 12.6 (1995), pp. 42–50.
- [142] Joost de Kruijff and Hans Weigand. "Ontologies for Commitment-Based Smart Contracts". In: Oct. 2017, pp. 383–398. isbn: 978-3-319-69458-0. doi: 10.1007/978-3-319-69459-7_26.
- [143] Joost de Kruijff and Hans Weigand. "Understanding the blockchain using enterprise ontology". In: *International Conference on Advanced Information Systems Engineering*. Springer. Cham, 2017, pp. 29–43.
- [144] Thomas Kühne. "Matters of (meta-) modeling". In: *Software & Systems Modeling* 5.4 (2006), pp. 369–385.
- [145] Tsung-Ting Kuo, Hugo Zavaleta Rojas, and Lucila Ohno-Machado. "Comparison of blockchain platforms: a systematic review and healthcare examples". In: *Journal of the American Medical Informatics Association* 26.5 (2019), pp. 462–478.
- [146] Jan Ladleif, Christian Friedow, and Mathias Weske. "An architecture for multi-chain business process choreographies". In: *International Conference on Business Information Systems*. Springer. 2020, pp. 184–196.
- [147] Jan Ladleif, Ingo Weber, and Mathias Weske. "External data monitoring using oracles in blockchain-based process execution". In: *International Conference on Business Process Management*. Springer. 2020, pp. 67–81.
- [148] Jan Ladleif, Ingo Weber, and Mathias Weske. "External Data Monitoring Using Oracles in Blockchain-Based Process Execution". In: *Business Process Management: Blockchain and Robotic Process Automation Forum*. Ed. by Aleksandre Asatiani et al. Cham: Springer International Publishing, 2020, pp. 67–81. isbn: 978-3-030-58779-6.
- [149] Jan Ladleif and Mathias Weske. "A Legal Interpretation of Choreography Models". In: *Business Process Management Workshops*. Ed. by Chiara Di Francescomarino, Remco Dijkman, and Uwe Zdun. Cham: Springer International Publishing, 2019, pp. 651–663. isbn: 978-3-030-37453-2.

- [150] Jan Ladleif and Mathias Weske. "A Unifying Model of Legal Smart Contracts". In: *Conceptual Modeling*. Ed. by Alberto H. F. Laender et al. Cham: Springer International Publishing, 2019, pp. 323–337. isbn: 978-3-030-33223-5.
- [151] Jan Ladleif and Mathias Weske. "Time in blockchain-based process execution". In: *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 2020, pp. 217–226.
- [152] Jan Ladleif, Mathias Weske, and Ingo Weber. "Modeling and Enforcing Blockchain-Based Choreographies". In: *Business Process Management*. Ed. by Thomas Hildebrandt et al. Cham: Springer International Publishing, 2019, pp. 69–85. isbn: 978-3-030-26619-6.
- [153] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine generals problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), pp. 382–401.
- [154] Jong-Hyouk Lee. "BIDaaS: blockchain based ID as a service". In: *IEEE Access* 6 (2018), pp. 2274–2278.
- [155] Ronald M. Lee. "A logic model for electronic contracting". In: *Decision Support Systems* 4.1 (1988), pp. 27–44. issn: 0167-9236. doi: [https://doi.org/10.1016/0167-9236\(88\)90096-6](https://doi.org/10.1016/0167-9236(88)90096-6). url: <http://www.sciencedirect.com/science/article/pii/0167923688900966>.
- [156] Karen Levy. "Book-Smart, Not Street-Smart: Blockchain-Based Smart Contracts and The Social Workings of Law". In: *Engaging Science, Technology, and Society* 3.0 (2017), pp. 1–15. issn: 2413-8053. doi: 10.17351/ests2017.107. url: <https://www.estsjournal.org/index.php/ests/article/view/107>.
- [157] K. Li et al. "Proof of Vote: A High-Performance Consensus Protocol Based on Vote Mechanism and Consortium Blockchain". In: *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, Dec. 2017, pp. 466–473. doi: 10.1109/HPCC-SmartCity-DSS.2017.61.
- [158] Wenting Li et al. "Securing proof-of-stake blockchain protocols". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Cham: Springer, 2017, pp. 297–315.
- [159] Sin Kuang Lo et al. "Reliability analysis for blockchain oracles". In: *Computers & Electrical Engineering* 83 (2020), p. 106582. issn: 0045-7906. doi: <https://doi.org/10.1016/j.compeleceng.2020.106582>. url: <https://www.sciencedirect.com/science/article/pii/S0045790619316179>.
- [160] Denivaldo Lopes et al. "Mapping specification in MDA: From theory to practice". In: *Interoperability of enterprise software and applications*. Springer, 2006, pp. 253–264.
- [161] Fabrice Lumineau, Wenqian Wang, and Oliver Schilke. "Blockchain Governance—A New Way of Organizing Collaborations?". In: *Organization Science* 32.2 (2021), pp. 500–521. doi: 10.1287/orsc.2020.1379. eprint: <https://doi.org/10.1287/orsc.2020.1379>. url: <https://doi.org/10.1287/orsc.2020.1379>.
- [162] Loi Luu et al. "A secure sharding protocol for open blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2016, pp. 17–30.
- [163] Loi Luu et al. "Demystifying Incentives in the Consensus Computer". In: *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. Denver, Colorado, USA: ACM, 2015, pp. 706–719. isbn: 978-1-4503-3832-5. doi: 10.1145/2810103.2813659. url: <http://doi.acm.org/10.1145/2810103.2813659>.

- [164] Loi Luu et al. "Making Smart Contracts Smarter". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. CCS '16*. Vienna, Austria: ACM, 2016, pp. 254–269. isbn: 978-1-4503-4139-4. doi: 10.1145/2976749.2978309. url: <http://doi.acm.org/10.1145/2976749.2978309>.
- [165] Orlenys López-Pintado et al. "Caterpillar: A business process execution engine on the Ethereum blockchain". In: *Software: Practice and Experience* 49.7 (2019), pp. 1162–1193. doi:<https://doi.org/10.1002/spe.2702>. eprint:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2702>. url: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2702>.
- [166] Daniel Macrinici, Cristian Cartoceanu, and Shang Gao. "Smart contract applications within blockchain technology: A systematic mapping study". In: *Telematics and Informatics* 35.8 (2018), pp. 2337–2354. issn: 0736-5853. doi: <https://doi.org/10.1016/j.tele.2018.10.004>. url: <http://www.sciencedirect.com/science/article/pii/S0736585318308013>.
- [167] D. Magazzeni, P. McBurney, and W. Nash. "Validation and Verification of Smart Contracts: A Research Agenda". In: *Computer* 50.9 (2017), pp. 50–57.
- [168] Haikel Magrahi et al. "NFB: A Protocol for Notarizing Files over the Blockchain". In: *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IFIP 2018. Paris, France, 2018, pp. 1–4. doi: 10.1109/NTMS.2018.8328740.
- [169] Kamran Mammadzada et al. "Blockchain Oracles: A Framework for Blockchain-Based Applications". In: *Business Process Management: Blockchain and Robotic Process Automation Forum*. Ed. by Aleksandre Asatiani et al. Cham: Springer International Publishing, 2020, pp. 19–34. isbn: 978-3-030-58779-6.
- [170] Bill Marino and Ari Juels. "Setting Standards for Altering and Undoing Smart Contracts". In: *Rule Technologies. Research, Tools, and Applications*. Ed. by Jose Julio Alferes et al. Cham: Springer International Publishing, 2016, pp. 151–166. isbn: 978-3-319-42019-6.
- [171] O. Marjanovic and Z. Milosevic. "Towards formal modeling of e-contracts". In: *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference*. 2001, pp. 59–68. doi: 10.1109/EDOC.2001.950423.
- [172] Will Martino and Steward Popejoy. "The Kadena Public Blockchain, Project Summay Whitepaper". In: (2017). eprint: <http://kadena.io/docs/KadenaPublic.pdf>.
- [173] Ivan Martinovic, Lucas Kello, and Ivo Slugavonic. *Blockchains for Governmental Services: Design Principles, Applications, and Case Studies*. Tech. rep. Oxford, United Kingdom, 2017.
- [174] Aaron K Massey et al. "Evaluating existing security and privacy requirements for legal compliance". In: *Requirements engineering* 15.1 (2010), pp. 119–137.
- [175] Frano Škopljanač Mačina and Bruno Blaškovič. "Formal Concept Analysis, Overview and Applications". In: *Procedia Engineering* 69 (2014). Ed. by Bernhard Ganter, Gerd Stumme, and Rudolf Wille. 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013, pp. 1258–1267. issn: 1877-7058. doi: <https://doi.org/10.1016/j.proeng.2014.03.117>. url: <http://www.sciencedirect.com/science/article/pii/S1877705814003634>.
- [176] Jan Mendling et al. "Blockchains for Business Process Management - Challenges and Opportunities". In: *ACM Trans. Manage. Inf. Syst.* 9.1 (Feb. 2018), 4:1–4:16. issn: 2158-656X. doi: 10.1145/3183367. url: <http://doi.acm.org/10.1145/3183367>.
- [177] Weizhi Meng et al. "When intrusion detection meets blockchain technology: a review". In: *Ieee Access* 6 (2018), pp. 10179–10188.
- [178] Dmitry Meshkov, Alexander Chepuronoy, and Marc Jansen. "Short Paper: Revisiting Difficulty Control for Blockchain Systems". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Cham: Springer, 2017, pp. 429–436.

- [179] Eliza Mik. "Smart contracts: terminology, technical limitations and real world complexity". In: *Law, Innovation and Technology* 9.2 (2017), pp. 269–300. doi: 10.1080/17579961.2017.1378468. eprint: <https://doi.org/10.1080/17579961.2017.1378468>. url: <https://doi.org/10.1080/17579961.2017.1378468>.
- [180] Andrew Miller et al. "Sprites: Payment channels that go faster than lightning". In: *arXiv preprint arXiv:1702.05812* (2017).
- [181] David Mills et al. *Distributed ledger technology in payments, clearing, and settlement*. Tech. rep. Washington DC, DC, USA, 2016.
- [182] Z. Milosevic, S. Sadiq, and M. Orlowska. "Translating business contract into compliant business processes". In: *2006 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06)*. 2006, pp. 211–220.
- [183] Du Mingxiao et al. "A review on consensus algorithm of blockchain". In: *IEEE International Conference on Systems, Management and Cybernetics (SMC) 2017*. IEEE, 2017, pp. 2567–2572.
- [184] Carlos Molina-Jimenez, Santosh Shrivastava, and Massimo Strano. "A model for checking contractual compliance of business interactions". In: *IEEE Transactions on Services Computing* 5.2 (2011), pp. 276–289.
- [185] William Mougayar. *The business blockchain: promise, practice, and application of the next Internet technology*. John Wiley & Sons, 2016.
- [186] Sean. Murphy and Charley. Cooper. *Can smart contracts be legally binding contracts?* Tech. rep. London, United Kingdom: R3cev and Norton Rose Fulbright, June 2016.
- [187] Sean Murphy and Charley Cooper. *Can smart contracts be legally binding contracts?* Tech. rep. London, United Kingdom, 2016.
- [188] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: (2008).
- [189] Harish Natarajan, Karla Krause Solvej, and Luskin Gradstein Helen. *Distributed Ledger Technology (DLT) and Blockchain*. Tech. rep. Washington DC, DC, USA, 2017.
- [190] Alex Norta et al. *Self-aware agent-supported contract management on blockchains for legal accountability*. Tech. rep. Tallinn, Estonia, 2017.
- [191] Natalya F. Noy and Deborah L. McGuinness. *Ontology development 101: A guide to creating your first ontology*. Tech. rep. Stanford University, 2001.
- [192] Barack Obama. *Executive Order – Making Open and Machine Readable the New Default for Government Information*. url: <https://obamawhitehouse.archives.gov/the-press-office/2013/05/09/executive-order-making-open-and-machine-readable-new-default-government->. (accessed: 02.04.2020).
- [193] Jeroen van Oerle and Patrick Lemmens. *Distributed ledger technology for the financial industry*. Tech. rep. Amsterdam, The Netherlands, 2016.
- [194] Svein Ølnes, Jolien Ubacht, and Marijn Janssen. "Blockchain in government: Benefits and implications of distributed ledger technology for information sharing". In: *Government Information Quarterly* 34.11 (2017), pp. 355–364.
- [195] OMG. *Business Process Model and Notation*. Tech. rep. Needham, MA, USA: Object Management Group, 2015, p. 105.
- [196] OMG. *Object Constraint Language*. Tech. rep. Needham, MA, USA: Object Management Group, 2010, p. 105.
- [197] OMG. *Unified Modeling Language*. Tech. rep. Needham, MA, USA: Object Management Group, 2015, p. 105.
- [198] Wanda J. Orlikowski and Jack J. Baroudi. "Studying Information Technology in Organizations: Research Approaches and Assumptions". In: *Information Systems Research* 2.1 (1991), pp. 1–28. doi: 10.1287/isre.2.1.1. eprint: <https://doi.org/10.1287/isre.2.1.1>. url: <https://doi.org/10.1287/isre.2.1.1>.
- [199] Reggie O'Shields. "Smart contracts: Legal agreements for the blockchain". In: *NC Banking Inst.* 21 (2017), p. 177.

- [200] P. N. Otto and A. I. Anton. "Addressing Legal Requirements in Requirements Engineering". In: *15th IEEE International Requirements Engineering Conference (RE 2007)*. 2007, pp. 5–14.
- [201] M. Tamer Özsu and Patrick Valduriez, eds. *Principles of Distributed Database Systems*. 3rd ed. New York: Springer, 2011. isbn: 978-1-4419-8833-1. doi: 10.1007/978-1-4419-8834-8.
- [202] Michael Papazoglou. *Web services: principles and technology*. Pearson Education, 2008.
- [203] Mike P. Papazoglou and Benedikt Kratz. "A Business-Aware Web Services Transaction Model". In: *Service-Oriented Computing – ICSOC 2006*. Ed. by Asit Dan and Winfried Lamersdorf. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 352–364.
- [204] Rafael Pass and Elaine Shi. "The sleepy model of consensus". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Cham: Springer, 2017, pp. 380–409.
- [205] M. Pease, R. Shostak, and L. Lamport. "Reaching Agreement in the Presence of Faults". In: *J. ACM* 27.2 (Apr. 1980), pp. 228–234. issn: 0004-5411. doi: 10.1145/322186.322188. url: <http://doi.acm.org/10.1145/322186.322188>.
- [206] Ken Peffers et al. "A Design Science Research Methodology for Information Systems Research". In: *Journal of Management Information Systems* 24.3 (2007), pp. 45–77. doi: 10.2753/MIS0742-1222240302. eprint: <https://doi.org/10.2753/MIS0742-1222240302>. url: <https://doi.org/10.2753/MIS0742-1222240302>.
- [207] José Carlos Pereira. "The Genesis of the Revolution in Contract Law: Smart Legal Contracts". In: *Proceedings of the 12th International Conference on Theory and Practice of Electronic Governance*. ICEGOV2019. Melbourne, VIC, Australia: Association for Computing Machinery, 2019, 374–377. isbn: 9781450366441. doi: 10.1145/3326365.3326414. url: <https://doi.org/10.1145/3326365.3326414>.
- [208] Thierry Perroud and Reto Inversini. *Enterprise Architecture Patterns: Practical Solutions for Recurring IT-Architecture Problems*. Berlin: Springer, 2013. isbn: 978-3-642-37560-6. doi: 10.1007/978-3-642-37561-3.
- [209] Gareth W Peters and Efstathios Panayi. "Understanding Modern Banking Ledgers through Blockchain Technologies: Future of Transaction Processing and Smart Contracts on the Internet of Money". In: (2015). eprint: 1511.05740.
- [210] Felipe Pezoa et al. "Foundations of JSON Schema". In: *Proceedings of the 25th International Conference on World Wide Web*. WWW '16. Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee, 2016, 263–273. isbn: 9781450341431. doi: 10.1145/2872427.2883029. url: <https://doi.org/10.1145/2872427.2883029>.
- [211] Marc Pilkington. *Research Handbook on Digital Transformations*. Ed. by F Olleros Xavier, Majlinda Zhegu, and Edward Elgar. Cheltenham, United Kingdom: Edward Elgar Publishing, 2016, p. 225. isbn: 9781784717759.
- [212] George Pirlea and Ilya Sergey. "Mechanising blockchain consensus". In: *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*. ACM. New York, NY, USA: ACM, 2018, pp. 78–90.
- [213] Klaus Pohl. *Requirements engineering: fundamentals, principles, and techniques*. Springer Publishing Company, Incorporated, 2010.
- [214] D.S. Pradeepkumar et al. "Evaluating Complexity and Digitizability of Regulations and Contracts for a Blockchain Application Design". In: *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. Gothenburg, Sweden: IEEE Computer Society, June 2018, pp. 25–29. doi: https://doi.org/10.475/123_4.

- [215] Cristian Prisacariu and Gerardo Schneider. "A formal language for electronic contracts". In: *International Conference on Formal Methods for Open Object-Based Distributed Systems*. Springer. 2007, pp. 174–189.
- [216] J Reinecke, G. Dessler, and W. Schoell. *Introduction to Business - A Contemporary View*. Boston, USA: Allyn and Bacon, 1989. isbn: 0205118321.
- [217] Olivier Rikken et al. *Smart Contracts as a specific application of blockchain technology*. Tech. rep. The Hague, The Netherlands, 2017.
- [218] Horst WJ Rittel and Melvin M Webber. "Dilemmas in a general theory of planning". In: *Policy sciences* 4.2 (1973), pp. 155–169.
- [219] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. "CoinShuffle: Practical decentralized coin mixing for Bitcoin". In: *European Symposium on Research in Computer Security*. Springer. 2014, pp. 345–364.
- [220] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The unified modeling language*. Tech. rep. Rational Software Corporation, 1999.
- [221] Ravi S Sandhu. "Role-based access control". In: *Advances in computers*. Vol. 46. Elsevier, 1998, pp. 237–286.
- [222] Ravi S Sandhu and Pierangela Samarati. "Access control: principle and practice". In: *IEEE communications magazine* 32.9 (1994), pp. 40–48.
- [223] Alexander Savelyev. "Contract law 2.0: 'Smart' contracts as the beginning of the end of classic contract law". In: *Information & Communications Technology Law* 26.2 (2017), pp. 116–134. doi: 10 . 1080 / 13600834 . 2017 . 1301036. eprint: <https://doi.org/10.1080/13600834.2017.1301036>. url: <https://doi.org/10.1080/13600834.2017.1301036>.
- [224] Donald J. Schepker et al. "The Many Futures of Contracts: Moving Beyond Structure and Safeguarding to Coordination and Adaptation". In: *Journal of Management* 40.1 (2014), pp. 193–225. doi: 10 . 1177 / 0149206313491289. eprint: <https://doi.org/10.1177/0149206313491289>. url: <https://doi.org/10.1177/0149206313491289>.
- [225] Jonas Schiffel et al. "Towards Correct Smart Contracts: A Case Study on Formal Verification of Access Control". In: *Proceedings of the 26th ACM Symposium on Access Control Models and Technologies*. SACMAT '21. Virtual Event, Spain: Association for Computing Machinery, 2021, 125–130. isbn: 9781450383653. doi: 10.1145/3450569.3463574. url: <https://doi.org/10.1145/3450569.3463574>.
- [226] David Schwartz, Noah Youngs, Arthur Britto, et al. *The Ripple Protocol Consensus Algorithm*. Tech. rep. San Francisco, CA, USA, 2014.
- [227] Bran Selic. "A systematic approach to domain-specific language design using UML". In: *10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*. IEEE. 2007, pp. 2–9.
- [228] Shane Sendall and Wojtek Kozaczynski. "Model transformation: The heart and soul of model-driven software development". In: *IEEE software* 20.5 (2003), pp. 42–45.
- [229] Mark D. Sheldon. "Auditing the Blockchain Oracle Problem". In: *Journal of Information Systems* 35.1 (June 2020), pp. 121–133. issn: 0888-7985. doi: 10 . 2308 / ISYS-19-049. eprint: <https://meridian.allenpress.com/jis/article-pdf/35/1/121/2815908/i1558-7959-35-1-121.pdf>. url: <https://doi.org/10.2308/ISYS-19-049>.
- [230] Andreas Sherborne. *Blockchain, Smart Contracts and Lawyers*. Tech. rep. London, United Kingdom, 2017.
- [231] Ric Shreves. *A Revolution in Trust: Distributed Ledger Technology in Relief and Development*. Tech. rep. Portland, OR ,USA, 2017.
- [232] Herbert A Simon. *The sciences of the artificial*. MIT press, 2019.

- [233] Si Alhir Sinan. *Understanding the Model Driven Architecture (MDA)*. url: <https://www.methodsandtools.com/archive/archive.php?id=5>. accessed: 19.07.2021.
- [234] Munindar P. Singh. "Norms as a Basis for Governing Sociotechnical Systems". In: *Transactions on Intelligent Systems and Technology* 5.1 (Jan. 2014). issn: 2157-6904. doi: 10.1145/2542182.2542203. url: <https://doi.org/10.1145/2542182.2542203>.
- [235] Yashwant Singh and Manu Sood. "Model Driven Architecture: A Perspective". In: *2009 IEEE International Advance Computing Conference*. 2009, pp. 1644–1652. doi: 10.1109/IADCC.2009.4809264.
- [236] Jeremy M Sklaroff. "Smart contracts and the cost of inflexibility". In: *University of Pennsylvania Law Review* 166 (2017), p. 263.
- [237] Amin Sleimi et al. "Automated extraction of semantic legal metadata using natural language processing". In: *2018 IEEE 26th International Requirements Engineering Conference (RE)*. IEEE. 2018, pp. 124–135.
- [238] *Smart Contracts Legal Agreements for the Digital Age*. Tech. rep. London, United Kingdom, 2018.
- [239] Michael Smith et al. *OWL 2 Web Ontology Language Conformance (Second Edition): W3C Recommendation 11 December 2012*. Tech. rep. The World Wide Web Consortium (W3C), 2012.
- [240] Yonatan Sompolinsky and Aviv Zohar. "Secure high-rate transaction processing in bitcoin". In: *International Conference on Financial Cryptography and Data Security*. Cham: Springer, 2015, pp. 507–527.
- [241] Victor Amaral de Sousa, Corentin Burnay, and Monique Snoeck. "B-MERODE: A Model-Driven Engineering and Artifact-Centric Approach to Generate Blockchain-Based Information Systems". In: *Advanced Information Systems Engineering*. Ed. by Schahram Dustdar et al. Cham: Springer International Publishing, 2020, pp. 117–133. isbn: 978-3-030-49435-3.
- [242] Stark. *Making Sense of Blockchain Smart Contracts*. 2016. url: <https://www.coindesk.com/making-sense-smart-contracts>.
- [243] Josh Stark. *Applications of Distributed Ledger Technology to Regulatory and Compliance Processes*. Tech. rep. New York, NY, USA, 2017.
- [244] Melanie Swan. *Blockchain: Blueprint for a new economy*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2015. isbn: 978-1491920497.
- [245] Nick Szabo. "Formalizing and securing relationships on public networks". In: *First Monday* 2.9 (1997).
- [246] Bayu Adhi Tama et al. "A critical review of blockchain and its current applications". In: *International Conference on Electrical Engineering and Computer Science (ICECOS)* 2017. IEEE, 2017, pp. 109–113.
- [247] Don Tapscott and Alex Tapscott. *Realizing the Potential of Blockchain, A Multi Stakeholder Approach to the Stewardship of Blockchain and Cryptocurrencies*. Tech. rep. Geneva, Switzerland, 2017.
- [248] Vyper Development Team. *Vyper Documentation*. <https://vyper.readthedocs.io/en/latest/index.html>. Online; accessed 18 June 2020. 2013.
- [249] Eric Tjong Tjin Tai. "Force majeure and excuses in smart contracts". In: (2018).
- [250] An Binh Tran, Qinghua Lu, and Ingo Weber. "Lorikeet: A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management." In: *CEUR Workshop (2018)*. 2018, pp. 56–60.
- [251] Guenter Heinz Treitel. *The law of contract*. Sweet & maxwell, 2003.

- [252] Florian Tschorsch and Björn Scheuermann. "Bitcoin and beyond: A technical survey on decentralized digital currencies". In: *IEEE Communications Surveys & Tutorials* 18.3 (2016), pp. 2084–2123.
- [253] Anna Vacca et al. "A systematic literature review of blockchain and smart contract development: Techniques, tools, and open challenges". In: *Journal of Systems and Software* 174 (2021), p. 110891. issn: 0164-1212. doi: <https://doi.org/10.1016/j.jss.2020.110891>. url: <https://www.sciencedirect.com/science/article/pii/S0164121220302818>.
- [254] Luuc Van Der Horst, Kim-Kwang Raymond Choo, and Nhien-An Le-Khac. "Process Memory Investigation of the Bitcoin Clients Electrum and Bitcoin Core". In: *IEEE Access* 5 (2017), pp. 22385–22398.
- [255] Arie Van Deursen and Paul Klint. "Domain-specific language design requires feature descriptions". In: *Journal of computing and information technology* 10.1 (2002), pp. 1–17.
- [256] Joan Verdon. *Global E-Commerce Sales To Hit 4.2 Trillion As Online Surge Continues*. url: <https://www.forbes.com/sites/joanverdon/2021/04/27/global-e-commerce-sales-to-hit-42-trillion-as-online-surge-continues-adobe-reports/?sh=194d125050fd>.
- [257] Friedhelm Victor and Bianca Katharina Lüders. "Measuring Ethereum-Based ERC20 Token Networks". In: *Financial Cryptography and Data Security*. Ed. by Ian Goldberg and Tyler Moore. Cham: Springer International Publishing, 2019, pp. 113–129. isbn: 978-3-030-32101-7.
- [258] K. Vidyasankar, P. Radha Krishna, and Kamalakara Karlapalem. "Study of Execution Centric Payment Issues in E-contracts". In: *2008 IEEE International Conference on Services Computing*. Vol. 2. 2008, pp. 135–142. doi: 10.1109/SCC.2008.40.
- [259] Aldenio de Vilaca Burgos et al. *Distributed ledger technical research in Central Bank of Brazil*. Tech. rep. Brasilia - DF, Brazil, 2017.
- [260] R Hevner Von Alan et al. "Design science in information systems research". In: *MIS quarterly* 28.1 (2004), pp. 75–105.
- [261] Mark Walport. *Distributed ledger technology: Beyond Blockchain*. Tech. rep. London, United Kingdom, 2016.
- [262] S. Wang et al. "Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49.11 (2019), pp. 2266–2277.
- [263] Yingli Wang, Jeong Hugh Han, and Paul Beynon-Davies. "Understanding blockchain technology for future supply chains: A systematic literature review and research agenda". In: *Supply Chain Management: An International Journal* 24.1 (2019), pp. 62–84.
- [264] Jos B Warmer and Anneke G Kleppe. *The object constraint language: getting your models ready for MDA*. Addison-Wesley Professional, 2003.
- [265] Ingo Weber et al. "Untrusted Business Process Monitoring and Execution Using Blockchain". In: *Business Process Management*. Ed. by Marcello La Rosa, Peter Loos, and Oscar Pastor. Cham: Springer International Publishing, 2016, pp. 329–347. isbn: 978-3-319-45348-4.
- [266] Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014.
- [267] Rolf T Wigand. "Electronic commerce: Definition, theory, and context". In: *The information society* 13.1 (1997), pp. 1–16.
- [268] Oliver E. Williamson. "Transaction-Cost Economics: The Governance of Contractual Relations". In: *The Journal of Law and Economics* 22.2 (1979), pp. 233–261. doi: 10.

- 1086/466942. eprint: <https://doi.org/10.1086/466942>. url: <https://doi.org/10.1086/466942>.
- [269] Claes Wohlin et al. *Experimentation in software engineering*. Springer Science & Business Media, 2012.
 - [270] Aaron Wright et al. *Smart Contracts & Legal Enforceability*. Tech. rep. New York, NY, United States, 2018.
 - [271] G. H. von Wright. "Deontic Logic". In: *Mind* 60.237 (1951), pp. 1–15. issn: 00264423, 14602113. url: <http://www.jstor.org/stable/2251395>.
 - [272] Wüst and Arthur Gervais. "Do you need a Blockchain?" In: (2017). doi: 20180828 : 104906. eprint: <https://eprint.iacr.org/2017/375>.
 - [273] M. Wöhler and U. Zdun. "Domain Specific Language for Smart Contract Development". In: *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 2020, pp. 1–9. doi: 10.1109/ICBC48266.2020.9169399.
 - [274] X. Xu et al. "A Taxonomy of Blockchain-Based Systems for Architecture Design". In: *Proceedings of the 2017 IEEE International Conference on Software Architecture (ICSA)*. Gothenburg: IEEE, Apr. 2017, pp. 243–252. doi: 10.1109/ICSA.2017.33.
 - [275] Xiwei Xu, Ingo Weber, and Mark Staples. "Introduction". In: *Architecture for Blockchain Applications*. Cham: Springer International Publishing, 2019, pp. 3–25. isbn: 978-3-030-03035-3. doi: 10.1007/978-3-030-03035-3_1. url: https://doi.org/10.1007/978-3-030-03035-3_1.
 - [276] Xiwei Xu et al. "A Pattern Collection for Blockchain-based Applications". In: *Proceedings of the 23rd European Conference on Pattern Languages of Programs*. ACM. 2018, p. 3.
 - [277] Xiwei Xu et al. "The blockchain as a software connector". In: *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. IEEE, 2016, pp. 182–191.
 - [278] Dylan Yaga et al. *Blockchain technology overview*. Tech. rep. Gaithersburg, MA, USA, 2018.
 - [279] Kazuhiro Yamashita et al. "Potential Risks of Hyperledger Fabric Smart Contracts". In: *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. 2019, pp. 1–10. doi: 10.1109/IWBOSE.2019.8666486.
 - [280] Wei Yin et al. "An anti-quantum transaction authentication approach in Blockchain". In: *IEEE Access* 6 (2018), pp. 5393–5401.
 - [281] Jesse Yli-Huuma et al. "Where Is Current Research on Blockchain Technology? A Systematic Review". In: *PLOS ONE* 11.10 (Oct. 2016), pp. 1–27. doi: 10.1371/journal.pone.0163477. url: <https://doi.org/10.1371/journal.pone.0163477>.
 - [282] Fei Richard Yu et al. "Virtualization for Distributed Ledger Technology (vDLT)". In: *IEEE Access* 6 (2018), pp. 25019–25028.
 - [283] Xiao Yu, Yueting Chai, and Yi Liu. "A Secure Model for Electronic Contract Enactment, Monitoring and Management". In: *2009 Second International Symposium on Electronic Commerce and Security*. Vol. 1. 2009, pp. 296–300. doi: 10.1109/ISECS.2009.184.
 - [284] Fan Zhang et al. "Town crier: An authenticated data feed for smart contracts". In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. ACM. 2016, pp. 270–282.
 - [285] Z. Zheng et al. "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends". In: *IEEE International Congress on Big Data (BigData Congress) 2017*. Honolulu: IEEE, June 2017, pp. 557–564. doi: 10.1109/BigDataCongress.2017.85.
 - [286] Y. Zhu et al. "TA-SPESC: Toward Asset-Driven Smart Contract Language Supporting Ownership Transaction and Rule-Based Generation on Blockchain". In: *IEEE Transactions on Reliability* (2021), pp. 1–16. doi: 10.1109/TR.2021.3054617.

-
- [287] Aviv Zohar. "Bitcoin: under the hood". In: *Communications of the ACM* 58.9 (2015), pp. 104–113.
 - [288] W. Zou et al. "Smart Contract Development: Challenges and Opportunities". In: *IEEE Transactions on Software Engineering* (2019), pp. 1–1. doi: 10 . 1109 / TSE . 2019 . 2942301.

CENTER DISSERTATION SERIES


CentER for Economic Research, Tilburg University, the Netherlands

| No. | Author | Title | ISBN | Published |
|-----|---------------------|---|-------------------|---------------|
| 638 | Pranav Desai | Essays in Corporate Finance and Innovation | 978 90 5668 639 0 | January 2021 |
| 639 | Kristy Jansen | Essays on Institutional Investors, Asset Allocation Decisions, and Asset Prices | 978 90 5668 640 6 | January 2021 |
| 640 | Riley Badenbroek | Interior Point Methods and Simulated Annealing for Nonsymmetric Conic Optimization | 978 90 5668 641 3 | February 2021 |
| 641 | Stephanie Koornneef | It's about time: Essays on temporal anchoring devices | 978 90 5668 642 0 | February 2021 |
| 642 | Vilma Chila | Knowledge Dynamics in Employee Entrepreneurship: Implications for parents and offspring | 978 90 5668 643 7 | March 2021 |
| 643 | Minke Remmerswaal | Essays on Financial Incentives in the Dutch Healthcare System | 978 90 5668 644 4 | July 2021 |
| 644 | Tse-Min Wang | Voluntary Contributions to Public Goods: A multi-disciplinary examination of prosocial behavior and its antecedents | 978 90 5668 645 1 | March 2021 |
| 645 | Manwei Liu | Interdependent individuals: how aggregation, observation, and persuasion affect economic behavior and judgment | 978 90 5668 646 8 | March 2021 |
| 646 | Nick Bombaïj | Effectiveness of Loyalty Programs | 978 90 5668 647 5 | April 2021 |
| 647 | Xiaoyu Wang | Essays in Microeconomics Theory | 978 90 5668 648 2 | April 2021 |
| 648 | Thijs Brouwer | Essays on Behavioral Responses to Dishonest and Anti-Social Decision-Making | 978 90 5668 649 9 | May 2021 |
| 649 | Yadi Yang | Experiments on hold-up problem and delegation | 978 90 5668 650 5 | May 2021 |
| 650 | Tao Han | Imperfect information in firm growth strategy: Three essays on M&A and FDI activities | 978 90 5668 651 2 | June 2021 |

| No. | Author | Title | ISBN | Published |
|-----|------------------------|---|-------------------|----------------|
| 651 | Johan Bonekamp | Studies on labour supply, spending and saving before and after retirement | 978 90 5668 652 9 | June 2021 |
| 652 | Hugo van Buggenum | Banks and Financial Markets in Microfounded Models of Money | 978 90 5668 653 6 | August 2021 |
| 653 | Arthur Beddock | Asset Pricing with Heterogeneous Agents and Non-normal Return Distributions | 978 90 5668 654 3 | September 2021 |
| 654 | Mirron Adriana Boomsma | On the transition to a sustainable economy: Field experimental evidence on behavioral interventions | 978 90 5668 655 0 | September 2021 |
| 655 | Roweno Heijmans | On Environmental Externalities and Global Games | 978 90 5668 656 7 | August 2021 |
| 656 | Lenka Fiala | Essays in the economics of education | 978 90 5668 657 4 | September 2021 |
| 657 | Yuxin Li | Pricing Art: Returns, Trust, and Crises | 978 90 5668 658 1 | September 2021 |
| 658 | Ernst Roos | Robust Approaches for Optimization Problems with Convex Uncertainty | 978 90 5668 659 8 | September 2021 |
| 659 | Joren Koëter | Essays on asset pricing, investor preferences and derivative markets | 978 90 5668 660 4 | September 2021 |
| 660 | Ricardo Barahona | Investor Behavior and Financial Markets | 978 90 5668 661 1 | October 2021 |
| 660 | Stefan ten Eikelder | Biologically-based radiation therapy planning and adjustable robust optimization | 978 90 5668 662 8 | October 2021 |
| 661 | Maciej Husiatyński | Three essays on Individual Behavior and New Technologies | 978 90 5668 663 5 | October 2021 |
| 662 | Hasan Apakan | Essays on Two-Dimensional Signaling Games | 978 90 5668 664 2 | October 2021 |
| 663 | Ana Moura | Essays in Health Economics | 978 90 5668 665 9 | November 2021 |
| 664 | Frederik Verplancke | Essays on Corporate Finance: Insights on Aspects of the General Business Environment | 978 90 5668 666 6 | October 2021 |

| No. | Author | Title | ISBN | Published |
|-----|--------------------|---|-------------------|---------------|
| 665 | Zhaneta Tancheva | Essays on Macro-Finance and Market Anomalies | 978 90 5668 667 3 | November 2021 |
| 666 | Claudio Baccianti | Essays in Economic Growth and Climate Policy | 978 90 5668 668 0 | November 2021 |
| 667 | Hongwei Zhang | Empirical Asset Pricing and Ensemble Machine Learning | 978 90 5668 669 7 | November 2021 |
| 668 | Bart van der Burgt | Splitsing in de Wet op de vennootschapsbelasting 1969 Een evaluatie van de Nederlandse winstbelastingregels voor splitsingen ten aanzien van lichamen | 978 90 5668 670 3 | December 2021 |
| 669 | Martin Kapons | Essays on Capital Markets Research in Accounting | 978 90 5668 671 0 | December 2021 |
| 670 | Xolani Nghona | From one dominant growth mode to another: Switching between strategic expansion modes | 978 90 5668 672 7 | December 2021 |
| 671 | Yang Ding | Antecedents and Implications of Legacy Divestitures | 978 90 5668 673 4 | December 2021 |
| 672 | Joobin Ordoobody | The Interplay of Structural and Individual Characteristics | 978 90 5668 674 1 | February 2022 |
| 673 | Lucas Avezum | Essays on Bank Regulation and Supervision | 978 90 5668 675 8 | March 2022 |
| 674 | Oliver Wichert | Unit-Root Tests in High-Dimensional Panels | 978 90 5668 676 5 | April 2022 |
| 675 | Martijn de Vries | Theoretical Asset Pricing under Behavioral Decision Making | 978 90 5668 677 2 | June 2022 |
| 676 | Hanan Ahmed | Extreme Value Statistics using Related Variables | 978 90 5668 678 9 | June 2022 |
| 677 | Jan Paulick | Financial Market Information Infrastructures: Essays on Liquidity, Participant Behavior, and Information Extraction | 978 90 5668 679 6 | June 2022 |
| 678 | Freek van Gils | Essays on Social Media and Democracy | 978 90 5668 680 2 | June 2022 |

| No. | Author | Title | ISBN | Published |
|-----|------------------|---|-------------------|----------------|
| 679 | Suzanne Bies | Examining the Effectiveness of Activation Techniques on Consumer Behavior in Temporary Loyalty Programs | 978 90 5668 681 9 | July 2022 |
| 680 | Qinnan Ruan | Management Control Systems and Ethical Decision Making | 978 90 5668 682 6 | June 2022 |
| 681 | Lingbo Shen | Essays on Behavioral Finance and Corporate Finance | 978 90 5668 683 3 | August 2022 |
| 682 | Joshua Eckblad | Mind the Gales: An Attention-Based View of Startup Investment Arms | 978 90 5668 684 0 | August 2022 |
| 683 | Rafael Greminger | Essays on Consumer Search | 978 90 5668 685 7 | August 2022 |
| 684 | Suraj Upadhyay | Essay on policies to curb rising healthcare expenditures | 978 90 5668 686 4 | September 2022 |
| 685 | Bert-Jan Butijn | From Legal Contracts to Smart Contracts and Back Again: An Automated Approach | 978 90 5668 687 1 | September 2022 |



ISBN: 978 905668 687 1
DOI: 10.26116/d6h4-7r79