DOCTORAL THESIS

A Legally Relevant Socio-Technical Language Development for Smart Contracts

Vimal Kumar Dwivedi

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2022

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
14/2022

A Legally Relevant Socio-Technical Language Development for Smart Contracts

VIMAL KUMAR DWIVEDI

TALTECH PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

This dissertation was accepted for the defence of the degree of Doctor of Philosophy (Informatics) on 18 April 2022

Defence of the thesis: 25 May 2022, Tallinn

Declaration:
I hereby declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.

Vimal Kumar Dwivedi

____
signature

Copyright: Vimal Kumar Dwivedi, 2022
ISSN 2585-6898 (publication)
ISBN 978-9949-83-815-8 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9949-83-816-5 (PDF)
Printed by Koopia Niini & Rauam

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
14/2022

Arukate lepingute jaoks õiguslikult asjakohane sotsiaal-tehniline keelearendus

VIMAL KUMAR DWIVEDI

Contents

List of Publications

This Ph.D. thesis is based on the following publications that are referred to in the text by Roman numbers.

I V. Dwivedi, V. Pattanaik, V. Deval, A. Dixit, A. Norta, and D. Draheim. Legally enforceable smart-contract languages: A systematic literature review. *ACM Comput. Surv.*, 54(5), June 2021

II V. Dwivedi, A. Norta, A. Wulf, B. Leiding, S. Saxena, and C. Udokwu. A formal specification smart-contract language for legally binding decentralized autonomous organizations. *IEEE Access*, 9:76069–76082, 2021

III V. Dwivedi and A. Norta. A legal-relationship establishment in smart contracts: Ontological semantics for programming-language development. In M. Singh, V. Tyagi, P. K. Gupta, J. Flusser, T. Ören, and V. R. Sonawane, editors, *Advances in Computing and Data Sciences*, pages 660–676, Cham, 2021. Springer International Publishing

IV V. Dwivedi and A. Norta. Auto-generation of smart contracts from a domain-specific xml-based language. In S. C. Satapathy, P. Peer, J. Tang, V. Bhateja, and A. Ghosh, editors, *Intelligent Data Engineering and Analytics*, pages 549–564, Singapore, 2022. Springer Singapore

V V. K. Dwivedi and A. Norta. A legally relevant socio-technical language development for smart contracts. In *Proceedings - 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems, FAS*W 2018*, pages 11–13. Institute of Electrical and Electronics Engineers Inc., 2018

VI V. Dwivedi, V. Deval, A. Dixit, and A. Norta. Formal-Verification of Smart-Contract Languages: A Survey. In *Advances in Computing and Data Sciences*, pages 738–747. Springer Singapore, 2019

Author's Contributions to the Publications

I In I, I was the primary author, and was in charge of interpreting the findings, preparing the figures, and writing the article.

II In II, I was the primary author, and was involved in data collection and analysis as well as writing the article.

III In III, I was the primary author, and created the proposed platform's ontology and workflow model, as well as the figures and the article.

IV In IV, I was the primary author and reviewed relevant articles before developing the model verification method, analyzing and interpreting the results, preparing the figures, and writing the article.

V In V, I was the primary author, and created the new XML-based smart-contract language, as well as preparing the figures and writing the article.

VI In VI, I was the primary author, and I proposed the transformation rules for converting an XML-based smart contract to a choreography model, as well as preparing the figures and writing the article.

Abbreviations

ADICO	Attributes, Deontics, Aims, Conditions, Or-else
ABM	Agent-based Models
AOM	Agent Oriented Modeling
BCRL	Business-Level Rules Language
BitML	Bitcoin Modeling Language
BNM	Business Network Model
BPaaS	Business Processes as a Service
BPMN	Business Process Modeling Notation
CarMan	Car Manufacturer
CC	Conventional Contract
CML	Contract Modeling Language
CPN	Colored Petri Nets
DAML	Digital Asset Modeling Language
DAO	Decentralized Autonomous Organization
DeFi	Decentralized Finance
DLT	Distributed Ledger Technology
DSL	Domain-Specific Language
DSR	Design Science Research
EOS	Electro-Optical System
eSML	eSourcing Markup language
EVM	Ethereum Virtual Machine
FSM	Finite State Machine
FSIS	Food Safety Information System
FSQAS	Food Safety and Quality Assurance System
IOC	Inter-Organizational Collaboration
IOT	Internet of Things
IR	Intermediate Representation
IS	Information System
IT	Information Technology
LLL	Lisp-Like Language
MAS	Multi Agent System
OCL	Object-Constraint Language
OEM	Original Equipment Manufacturer
OOP	Object-Oriented Programming
P2P	Peer-to-Peer
PK	Public Key
PKC	Public Ket Cryptography
PN	Petri-Net
PoS	Proof of Stake
PoW	Proof of Work
QSCL	Qtum Smart-contract Language
RQ	Research Question
S/W	Strength and Weakness
SC	Smart Contract
SCL	Smart-contract Language
SCM	Supply Chain Management
SLCML	Smart Legal Contract Markup Language

SLR	Systematic Literature Review
SmaCoNat	Smart Contracts in Natural Language
SPESC	A Specification Language for Smart Contracts
SOC	Service Oriented Computing
STS	Socio-Technical System
TON	The Open Network
UML	Universal Modeling Language
UT	Unit Test
VE	Virtual Enterprise
VM	Virtual Machine
VTP	Value Transfer Protocol
WES	Workflow Enactment Service
XML	Extensible Markup Language

1 INTRODUCTION

1.1 Thesis Motivation

Many organizations consider collaboration with other companies and organizations in their supply chains to be critical for survival in modern hyper-competitive environments. For the purpose of this thesis, this collaboration is referred to as inter-organizational collaboration (IOC). Organizations may collaborate with others to develop innovative products or services [139], resolve impressive challenges [149], standardize their supply chains [57], set and manage standards [88], work on research and development projects [173], or re-spond to emergencies [13]. By collaborating with other companies, businesses can pool resources and achieve goals [71] in ways they could not do so independently.

Many businesses use workflow enactment services (WES) to automate and streamline their business processes to improve efficiencies, reduce costs, and increase their profitability [138]. Although each organization is unique, many of the work processes are common across all organizations within a specific industry, and as a result, workflow enactment services are designed in such a way they can be configured to meet the needs of a particular organization. In this way WESs are equipped to handle common business processes in many different types of organizations, and across industries. When selecting a WES, an organization typically looks for a workflow vendor who understands their business, and who is able to meet their specific needs.

Any form of collaboration, whether between people or organizations, requires communication and cooperation. When two or more organizations collaborate, multiple business processes are involved. Each organization is a system in its own right; with their own processes and workflows. Effective collaboration requires that the WES of each of the collaborating organizations are able to communicate and cooperate with one another. Information Technology (IT) professionals have long struggled to preserve consistency and mutual trust in inter-organizational business processes [1, 86]. Information on business operations can be shared and validated within an organization's centralized business processes where the process participants trust one another. When control over a process is delegated outside of an organization, as in an inter-organizational collaboration process, it is not possible for either organization to validate accuracy of data, enforce contractual obligations, or ensure that specific conditions are met. As a consequence, transferring control between fragile processes across organizations can lead to inconsistency and a lack of trust in process management [105].

Blockchain technology has the potential to significantly alter the business ecosystem, allowing for the execution, monitoring, and improvement of business processes within or across computer networks [110]. Blockchain is a distributed database technology based on a timestamped list of transaction records that cannot be tampered with [54]. Blockchains provide a secure way to execute processes in a trusted environment even in networks where nodes do not trust each other. This is aided by peer-to-peer networks, consensus mechanisms, cryptography techniques, and distributed ledger technology (DLT).

The secure nature of blockchain technology has also played an important role in the development and implementation of smart contracts (SC). As envisaged by its creator [90], a smart contract is a self-executing computer program designed to automate the sales process. The SC, which is stored in a distributed, decentralized blockchain system, has the terms and conditions of the agreement between buyer and seller written directly into the lines of code. The SC is activated automatically when predetermined conditions are met. Szabo's vision of an SC, which he defines as "a set of promises, specified in digital form, including protocols within which the parties perform on these promises" [159], has come to fruition as a result of the inherent characteristics of blockchain technology that have driven its implementation.

Business processes are governed by sets of rules that define how they respond to specific conditions. For example, assume a sales transaction in which a customer orders 200 goods from a seller. The terms and conditions of this simple contract specify a maximum delivery period of two weeks. If the seller fails to deliver within that period, the customer may be entitled to a penalty payment. Business rules like these can be expressed using SCs, and executed automatically by distributed and decentralized blockchain technology. It is this combination of blockchain technology and SCs that govern the execution of inter-organizational business processes, and enable the functioning of decentralized autonomous organizations (DAO) [124]. Solidity [1], Michelson [2], and Rholang [3] are among the new smart-contract languages (SCLs) that have emerged to implement executable SCs.

A DAO is a virtual enterprise (VE); a digital entity that exists only as code stored on multiple distributed and decentralized computers, networks and nodes. Unlike a traditional business, a DAO has no central leadership, rather it is organized around a specific set pf rules enforced on a blockchain [59, 54]. The rules that govern the operation of the DAO are encoded in computer programs that are transparent and open. The first DAO, created by developers as a new, decentralized business model for both commercial and non-profit enterprises, was written in open-source code, and was based on the Ethereum blockchain. It was designed as an investor-directed venture capital fund, in which people from anywhere in the world could invest money anonymously and receive cryptocurrency; ethereum tokens, to the value of their investment, in exchange. 'The DAO' was launched in April 2016 after a very successful worldwide crowdfunding campaign, in which more than US$ 150 million was generated from selling digital tokens [7]. Unfortunately, less than two months later, in June 2016, hackers were able to exploit vulnerabilities in the source code, and stole US$ 50 million [49]. These hackers exploited 'call to the unknown' and 're-entrancy' vulnerabilities; programming errors which were introduced because of the open-source software, and the collaborative nature of the programmers.

Although this was a tragic failure, it proved to be an important model for modern DAOs. 'The DAO' demonstrated how blockchain technology can be used to improve efficiency, effectiveness and quality of business processes in an automated, distributed and decentralized collaborative environment in which each organization contributes to a network of peers, governed by SCs that regulate and limit individual business behavior [54].

In the context of a DAO, each business is a self-contained, decentralized, and self-organizing network that allows for a more rapid and cost-effective response to market shifts. Business enterprises act as peers or agents that execute particular operations in a collaboration lifecycle involving both humans and software agents. The DAO uses peer-to-peer (P2P) computing without the use of clouds or servers in a loosely-coupled collaboration lifecycle in which software agents participate in the setup [121], enactment [123], potential rollback, and finally, orderly termination of smart contracts.

The collaboration lifecycle simplifies the selection and use of DAO-provided services, the negotiation of SCs, the monitoring of behavior during the enactment of the SC, and the management of any breach of contract by either party [129]. Unfortunately, despite the fact that blockchains are designed to provide a technological framework for drafting legally-binding SCs, the underlying contractual concepts and properties required to make SCs legally binding, referred to as 'suitability' [128]) are still understudied [31], [66], [68].

[1]Solidity| Docs Page
[2]Tezos| Home Page
[3]Rholang| GitHub Page

According to Norta et al., the ontological suitability of smart-contract languages (SCL) for the drafting of legally-binding SCs can be realized through (i) the choreography, or workflow, of processes [54] (referred to as 'concepts'), and (ii) the semantics that define the individual DAO processes (referred to as 'properties') [128].

Established SCLs (such as Solidity [39], Vyper [19], and others) are programming languages specifically designed for writing smart contracts. Unfortunately, due to their highly technical and and specialized nature, the SCs developed using these languages are not easily understood by non-IT practitioners. And even for IT specialists, assessing the legal requirements of SCs is challenging and time-consuming due to their lack of legal knowledge. While SCLs like Solidity and Vyper, have been developed specifically for the creation of smart contracts, and are designed to minimize coding mistakes, lack of alignment between SCL semantics and software developers can lead to errors being inadvertently introduced into the SC code, with serious financial implications and the potential for significant losses.

For the purpose of this thesis, lack of semantic alignment between SCL and human coders is referred to as 'expressiveness' [see Publication (I)], and the example cited previously when hackers exploited 'call to the unknown' and 're-entrancy' vulnerabilities, and stole US$ 50 million from 'The DAO' in June 2016 [48], is possibly the most well-known illustration of lack of expressiveness, or formal verification, in existing SCLs.

The legal properties of SCs, including the rights and obligations of the contracting parties, are analogous to software specifications the developer or programmer must address when coding. It is therefore essential for software developers and programmers to have sufficient legal knowledge to write contract content. Programmers also have to be able to communicate with business people in order to elicit and clearly define what the software is required to do and how it is expected to perform under practical business conditions.

As a result of the interdisciplinary nature of SCs, it is essential that computer scientists, software developers and programmers work collaboratively with lawyers, business and financial experts, and other specialists from the various domains within an organisation when proposing, designing and implementing legally-binding SCs.

Given that suitability and expressiveness SCL properties are required for drafting and enforcing formally-verifiable, legally-binding SCs in DAO collaborations, the overall goal of this thesis is to identify these properties and develop a novel SCL to support them.

It should be noted that although SCLs supporting legally binding concepts and properties of SCs have been discussed in the scientific literature, for example: Specification Language for Smart Contracts (SPESC) [75], Symboleo [152], and Smart Contracts in Natural Language (SmaCoNat) [140]. Each of these SCLs have their disadvantages; For instance, SPESC only focuses on modeling legal positions, or legal relationships , and not on other critical aspects of contracts such as: obligation states, or rights and obligations categories. Whereas Symboleo is flexible enough to capture a wide range of real-world contracts, it lacks the concepts and properties to address collaborative contracts. The SmaCoNat contract modeling language (CML) does not address transaction rules and is, therefore, inadequate for the formulation any type of business contracts, because it does not meet the requirements of domain completeness. Other SCLs and their drawbacks are discussed in detail in Publication (I). Despite the fact that SPESC, Symboleo and SmaCoNat present intriguing approaches for development of legally binding SCs, most, if not all, of these contract modeling languages either lack domain-completeness, or are intended for non-collaborative business processes.

This thesis addresses the shortcomings of existing SC modeling languages which are often inadequate for legal recognition, particularly in smart-contract-enabled funding rounds.

They are also not able to reflect the dynamics of collaborative business-processes.

Legally-binding and collaborative properties include indecomposable elements such as ‘semantic suitability’ (domain-completeness) and ‘workflow suitability’ (collaborative properties) [Publication (I)]. Semantic suitability provides insights into the context of a smart contract, for example: who the participants to the transaction are, what they are exchanging, and under what terms and conditions this exchange takes place [70]. Workflow suitability [146] embodies properties that provide insights into the processes and workflow patterns to be followed when conducting a smart contract from the perspective of contractual collaboration, for example: how the transactions are to be carried out, and the workflow model that will be applied. The aim of this thesis is the development of an ontology which addresses the semantic suitability properties of SCs [Publication (II)].

Many of the existing SCLs are unaware of their own processing state, and as a result, should not be classified as ‘smart’. When a conflict arises, tracing how the SC is executed is challenging, and because of this SCs are difficult to enforce legally. To address these issues, this thesis aims to develop a smart contract workflow model [see Publication (III)] in Colored Petri Nets (CPNs) which can be used to design, develop, and analyze the processing state of SCs in order to track the fulfillment of contractual properties [93].

This thesis also address the problem of alignment between SCL semantics and the intuition of human programmers and software developers, by developing a blockchain-independent formal specification language [Publication (II)]. This language will allow blockchain developers to focus on the contractual workflow rather than the specific syntax of each blockchain platform. Because formal-specification SCLs are typically not blockchain executable, an additional objective of this thesis is to propose a novel approach that would allow legally-binding SCs to be executed automatically on any blockchain platform [see Publication (IV)].

1.2 Research Questions

This thesis addresses three primary topics, and for each topic, a short description is provided, together with a number of research questions aimed at specific nuances of the topic. Table 1 outlines the relevant connections between research questions, thesis chapters, publications, and contributions (described in subsection 1.3 below).

Assessment of existing SCLs and identification of SCL properties that can render SCs legally binding It is unclear how to create an SCL capable of allowing for the creation of legally-binding SCs. Although existing SCLs such as Solidity, Michelson, and Rholang have been developed for implementing executable smart contracts, they lack the critical legal and contractual properties required for drafting and enforcing formally-verifiable and legally-binding smart contracts. Without these properties, it is not possible to define contracting concepts such as: who the parties to the transaction are, what the transaction is, what they are exchanging, and what specific provisions are included in the smart contract.

The first research question (RQ1) helps provide a better understanding of the current approaches for developing an SCL and proposes the suitability and expressiveness properties that are critical for developing legally binding SCLs. This RQ also aims to asses whether the existing SCLs have the required suitability and expressiveness properties necessary for developing DAO smart contracts. Based on the findings of this assessment, a novel framework is devised for designing SCLs that address the weaknesses of the current approaches.

RQ1 How can a novel framework be devised for designing smart contract languages that are semantically rich and which support the drafting of formally-verifiable smart contracts, for use in DAO collaboration?

In order to answer RQ1, three sub-questions are derived.

- SRQ1.1 What blockchain-based SCLs already exist in scientific and non-scientific literature?
- SRQ1.2 What properties of business-oriented SCLs contribute to suitability and expressiveness?
- SRQ1.3 What obstacles are there in existing SCLs that restrict the attainment of business contractual objectives?

Implementation of formal specification language The use of blockchain and smart contracts technology improves the efficiency and automation of business processes. The growing interest in developing decentralized autonomous organizations (DAOs) demonstrates that blockchain technology has the potential to transform business and society. However, due to a lack of ontologically legally-binding SCL concepts and properties, SCs are currently not able to capture the full range of business-related contracts.

The second research question (RQ2) describes the necessary set of concepts and properties, which are conceptualized in a multi-tiered SCL ontology that captures the full range of business-related contracts within a unified model. Furthermore, SCL ontology is formalized in Colored Petri Nets (CPNs) which can be used to design, develop, and analyze the processing state of SCs in order to track the fulfillment of contractual properties. Furthermore, the concepts and properties captured in the SCL ontology is translated into the extensible markup language (XML) i.e., smart-legal-contract markup language (SLCML).

- RQ2 How can a formal-specification language be developed for the purpose of legally-binding DAO collaboration?

  Hence, three sub-questions are derived to support RQ2.

- SRQ2.1 What formal semantics are required to define the legal aspects of a business process?
- SRQ2.2 What enactment mechanisms ensure the legal enforceability of contracts?
- SRQ2.3 What is the machine-readable language conversion based on the ontology?

Translation approach from specification language to blockchain-executable language Smart contracts written in existing languages, such as Solidity, Vyper, and others, are difficult for domain stakeholders and programmers to understand due to technical limitations inherent in the SCL. As a result of the conceptual gap between the contractual provisions required for legally-binding SCs and the limitations of the coding language, it is difficult for programmers to develop efficient, error-free code. Although formal specification languages such as SLCML, Symboleo, and others are introduced to address these issues, no translation approach has been implemented, or as yet proposed, to transform SCs written in a formal specification language into blockchain machine-readable code.

The third research question (RQ3) proposes a pattern, as well as transformation rules, for translating contracts written in SLCML to blockchain executable machine code in order to reduce the effort and risk associated with smart contract development. This is accomplished by implementing an SLCML code instantiation of examples of real-world contracts, after which, the proposed patterns and transformation rules are used to build a business process modeling notation (BPMN) choreography model from the SLCML-coded contracts.

RQ3 How can a BPMN choreography model be built to translate an XML-based contract to blockchain-executable language?

The following sub-questions are derived for RQ3.

SRQ3.1 What is the structure of the SLCML instantiation that is crucial for the choreography transformation?

SRQ3.2 What are the patterns and rules for converting SLCML code to a BPMN choreography model?

SRQ3.3 What is the feasibility-evaluation approach of the proposed solution for a use case?

Table 1: Mapping of research questions to corresponding thesis chapters, publications, and contributions.

Research Questions	Chapters	Publications	Contributions
SRQ1.1, SRQ1.2, SRQ1.3	3	I, VI	1, 2
SRQ2.1, SRQ2.2, SRQ2.3	4	II, III	3, 4, 5
SRQ3.1, SRQ3.2, SRQ3.3	5	VI	6

1.3 Contributions

This thesis is based on a collection of articles published in journals, and from international conferences. The thesis identifies and proposes suitability and expressiveness properties that are critical for developing legally-binding SCLs. Furthermore, existing SCLs are evaluated to verify if they have the necessary suitability and expressiveness properties. In addition, a multi-tiered ontology comprising concepts and properties for the design of legally-relevant collaborative SCs is proposed. Moreover, a smart legal contract markup language (SLCML) is developed to define the configuration (not the development) of a legally-binding SC. Finally, a pattern and associated transformation rules, are proposed for translating SLCML coded SCs to blockchain executable code.

The main contributions of this thesis are:

1. Recommendations for suitability and expressiveness properties. The thesis identifies critical properties such as semantic suitability, workflow suitability and expressiveness, which can render smart contracts legally enforceable.

2. Recommendations for the development of legally binding SCLs. The contract enabling properties and obstacles that restrict the achievement of business contractual objectives of 45 existing SCLs are evaluated and discussed. Based on this analysis, a novel framework for designing SCLs that are semantically rich and support the drafting of formally-verifiable smart contracts is proposed.

3. Development of multi-tiered SCL ontology. Business contracts can be of different types, and because the realm and range of each type differ greatly, it is difficult to

express the entire spectrum of contracts in a single ontology. This thesis proposes a multi-tiered SCL ontology that is semantically rich enough to configure the full range of business-related contracts.

4. Modeling of SC workflow. Since SCs are unaware of their own processing state, tracing the past performance of SC execution is difficult. More specifically, this thesis demonstrates how procedural knowledge about the expected flow of business actions within the business process workflow of the individual contracting parties is obtained.

5. Development of formal-specification smart-legal contract markup language (SLCML). The proposed novel smart-legal contract markup language contains the necessary vocabulary required to define legally-binding collaborative business contracts.

6. Translation approach from formal specification language to blockchain code. This thesis proposes a pattern, and associated transformation rules, for creating a BPMN choreography model from XML-based smart-legal contract markup language. It also shows how the BPMN choreography model is translated into blockchain machine-readable code.

1.4 Research Methodology and Methods

The methodology used in this thesis is Design Science Research (DSR). This model provides a rigorous framework for the creation and evaluation of an information systems (IS) artefact for the express purpose of improving the functional performance of the designed artefact [166]. Design science is a structured, systematic, creative, and iterative process for the collaborative development of new solutions to existing business problems. The traditional DSR model is based on three pillars: the environment, the IS research cycle, and a knowledge base. The first pillar, the environment pillar, consisting of three interrelated elements; people, organization and technology, provides input into the second pillar; the IS research cycle. The focus of this pillar is the developing, creating, building, assessing, evaluating, justifying, reviewing, and improving, of an IT artefact that addresses the specific user, or business need. The third pillar; the knowledge base, represents the foundation methodology, theories, frameworks, models and methods that are necessary inputs for developing the new artefact. For the DSR methodology to be successful, it needs inputs from both the environment and the knowledge base pillars. For the purpose of this thesis, the DSR model has been adapted from [166] and is depicted in Figure 1.

In the adapted DSR model, the environment pillar represents the interaction between programmers, end users, business professionals and legal experts within decentralised, distributed and autonomous organizations engaged in inter-organizational collaboration using blockchain-based smart contracts. To identify relevant problems relating to the topic of this thesis, a large number of interactions between programmers, end users, and business professionals from industry and government are observed. In addition, dozens of supply chain, construction, and energy-related contracts are analyzed. The relevance of this research is, therefore, to address problems that exist as a result of this interaction.

The IS research cycle process relevant to this research is the developing of legally-binding smart contracts in decentralized, distributed autonomous businesses, engaged in inter-organizational collaboration. Furthermore, this thesis addresses the research problem related to the implementing of legally-binding SCs in DAOs using blockchain technology.

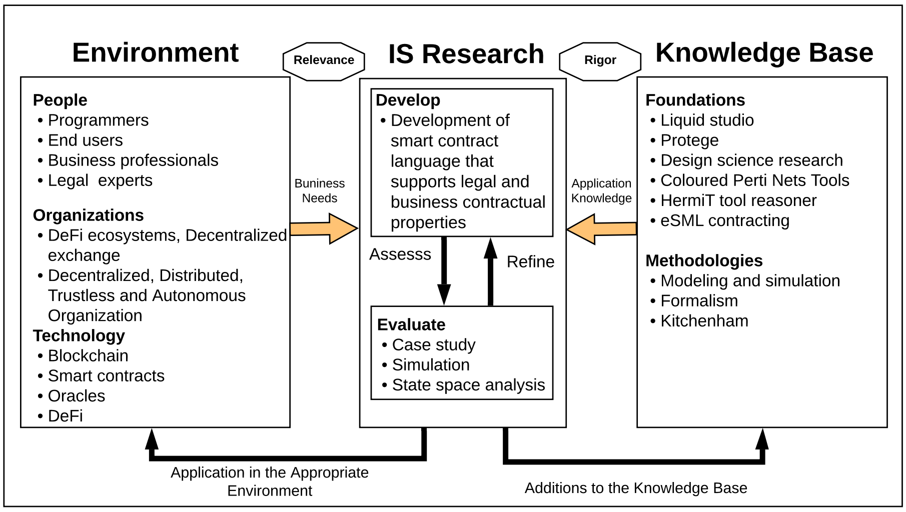
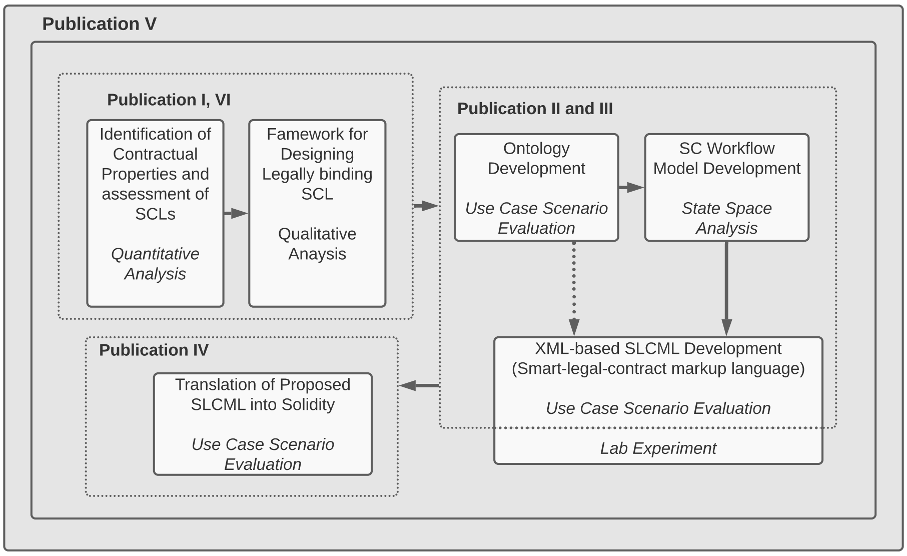


Figure 1: Design-science research method, adapted from [77].

The knowledge base provides useful models, methods, frameworks, and validation criteria for the artefacts created in this study; thus serving as the scientific foundation on which this study is based. In this thesis, a SLCML is developed as an artefact to support legal and business-contractual properties. A literature review is conducted (quantitative analysis), to identify existing SCLs in Publication (VI). The contribution of Publication (VI) is expanded in (I), in which a comprehensive meta-study of existing SCLs is conducted (quantitative analysis), to assess their contractual properties (see Figure 2). A framework is proposed through qualitative analysis, for developing legally-binding SCLs based on the findings of the meta-study. The Kitchenham methodology [85] is used for quantitative and qualitative analysis, because it is a transparent method for aggregating knowledge from available literature, and is based on an unbiased, auditable, and repeatable methodology. The framework proposed in Publication (I) is implemented as published in (II), (III), and (IV). The formal SLCML ontology is developed to conceptualize legal and business contractual properties. To analyze the processing state of contractual properties in SC execution, an SCL ontology is formalized in a colored petri net (CPN) tool, which is essentially a graphical SC workflow model. The concepts and properties of the SCL ontology are translated into machine-readable XML-based language. Patterns and transformation rules for translating SLCML contracts into Solidity are proposed (see Figure 2). The initial PhD project outline for this thesis is published in Publication (V). The subject of this thesis is the development and evaluation of an SLCML that supports legally-relevant business contractual properties.

The DSR method focuses on the creation and evaluation of information technology artefacts, as illustrated in Figure 1. For the purpose of this thesis, two evaluation methodologies: Use-case scenarios (running cases) and laboratory experiments, are used to demonstrate the generality and applicability of the artefacts in blockchain-based DAO collaboration. Using use-case scenarios, in the form of running cases, to demonstrate the generality of artefacts such as models, frameworks, ontologies and computer programs, is a well-accepted practice in blockchain research. As argued by Mendling et al., [111], arte-



Publication V
Publication I, VI
Identification of Contractual Properties and assessment of SCLs
Quantitative Analysis
Famework for Designing Legally binding SCL
Qualitative Anaysis
Publication II and III
Ontology Development
Use Case Scenario Evaluation
SC Workflow Model Development
State Space Analysis
XML-based SLCML Development (Smart-legal-contract markup language)
Use Case Scenario Evaluation
Lab Experiment
Publication IV
Translation of Proposed SLCML into Solidity
Use Case Scenario Evaluation

Figure 2: Illustration of contribution, research methodology and publication.

facts such as these "describe application scenarios involving blockchain technology in logistics and supply chain processes". Examples from literature, where running cases have been used to demonstrate blockchain-related technologies, include: agricultural supply chains [180], automotive supply chains [122], and food supply chains [15], among others. Keeping in line with this approach of utilizing use-case scenarios to demonstrate the generality of the created artefact, this work makes use of two running cases namely: an automotive supply chain case, and an agricultural supply chain case to evaluate the artefact.

These running cases provide reasonable coverage of the various kinds of contractual requirements and legal concerns, against which the artefact presented in this thesis is evaluated. Furthermore, the generality and applicability of the artefact are demonstrated through laboratory experiments conducted with participants who have blockchain expertise. During laboratory experiments the semantic qualities, and pragmatic usefulness of the SLCML is examined to determine its applicability within a collaborative inter-organizational DAO environment.

It is necessary to outline and explain how the research conducted in this thesis applies the principles of the DSR method. Table 2 presents the main guidelines of DSR and how they are applied in this thesis. The table shows the seven guidelines for conducting DSR and the description outlining how they are applied in this thesis.

Table 2: Guidelines of Design Science Research [166].

Guideline	Description
Design as an Artefact	It is essential that a DSR produces artefact(s). These artefacts can take the form of models, methods, or frameworks. An XML-based smart-legal contract markup language SLCML is the artefact developed for the research conducted in this thesis.
Problem Relevance	The DSR methodology focuses on addressing critical issues in the business environment. The relevant organizational problem addressed in this thesis is the absence of legal, and business-contractual concepts and properties in existing SCLs. Thus addressing the research question (RQ2) of how to develop a smart-contract language that has concepts and properties for guiding business collaboration in a legally relevant way.
Design Evaluation	The DSR methodology mandates that the artefact(s) created in any research that employs DSR methods be thoroughly evaluated. The SLCML developed in this thesis is evaluated through use-case scenarios (running cases) in order to demonstrate and understand the syntax and semantic properties of the SLCML schema. Furthermore, the usability of the SLCML is assessed through laboratory experiments to ensure its efficacy in developing legally binding SCs.
Research Contributions	The contributions of well-conducted DSR research must be clear. The main contribution of this thesis in the field of artefact design is the development of formal-specification smart-legal contract markup language (SLCML) that contains essential vocabularies to configure legally-binding and collaborative business contracts.
Research Rigor	This entails employing rigorous methods in the development and evaluation of DSR artefacts. Rigorosity is demonstrated in this thesis by developing the SLCML by extending the concepts and properties of the eSourcing markup language. The rigor in development of SLCML is demonstrated by analyzing various evaluation methods and adopting the most appropriate method to evaluate the developed artefacts.
Design as a search process	This guideline demonstrates that an effective artefact developed to address a specific organizational, or business, problem must also adhere to the rules of the problem environment. Thus, this thesis demonstrates that the developed SLCML artefact abides by the laws of the applicable domain, and the legal properties of the SLCML are evaluated by blockchain experts.
Communication of Research	This entails presenting the results of the research to audiences in the technology and management fields. The main findings of this thesis have been published in journals, and presented at several conferences, with audiences drawn from experts in technology and management-related fields.

The mapping of research methods to the corresponding research question and publications, is summarised in Table 3.

Table 3: Research methods used in this research.

RQ	Publications	Topic	Methods used
SRQ1.1, SRQ1.2, SRQ1.3.	I	SCL framework	Kitchenham methodology; literature review, data collection from scientific databases, data analysis
	VI	SCL properties	literature review, data analysis.
SRQ2.1, SRQ2.2, SRQ2.3.	II	Ontology and SLCML design	Design science research; case study, data collection from legal practitioners.
	III	Workflow model	Design science research; case study, simulation data collection from legal practitioners.
SRQ3.1, SRQ3.2, SRQ3.3.	IV	Translation approach	Design science research; case study (data collected from business case).

1.5 Structure of the Thesis

This thesis is divided into seven chapters. The introductory chapter provides a brief overview of the SCL in drafting legally binding SCs, as well as the research questions, and contribution of the thesis.

Chapter 2 provides a detailed background to this thesis by presenting literature reviews that serve as the foundation for the research. The running case(s) used to demonstrate the applicability and evaluation of the SCL developed in this thesis, are also included in the chapter.

Chapter 3 describes the critical properties that can aid in the creation of legally-binding SCs. This chapter also proposes a future research direction for developing legally enforceable SCLs.

Chapter 4 presents the novel semantically rich ontology for describing legally-binding concepts and properties for defining SCs, a workflow model (CPN model) for analyzing the processing state of SCs, and SLCML vocabularies for drafting legally binding collaborative SCs.

Chapter 5 discusses examples of SC code using the SLCML schema, proposes a translation approach from SLCML to blockchain-executable language, and discusses the feasibility of the SLCML-based SC translation approach to Solidity.

Chapter 6 discusses the existing evaluation approaches of modeling languages, identifies the best approach for evaluating SLCML, and discusses the evaluation results of SLCML language.

Chapter 7 provides the conclusion of the thesis by summarizing the main results of the study. Furthermore, the limitations and future work resulting from this thesis are presented.

2 BACKGROUND AND RELATED WORK

This chapter begins by introducing the fundamental concepts of blockchain technology before demonstrating the legal implications of blockchain smart-contracts. Section 2.1 provides a technical overview of SCLs after which the legal implications of SCs are explained in more detail. Section 2.2 discusses and compares related work, and expresses the motivation for this thesis. Section 2.3 outlines the smart-contract inter-organizational use-case scenarios that are used to demonstrate and evaluate the applicability of the SLCML developed in this thesis.

2.1 Blockchain and Smart Contracts

Blockchain technology was first introduced in 2009 with the Bitcoin cryptocurrency [119]. In this first use-case, a blockchain's decentralized nature enabled the transfer of cryptocurrency without the involvement of a central banking or financial authority, and thereby eliminated the cost of bank fees, taxes, and other intermediary expenses during transactions. A blockchain is an immutable distributed ledger that can be used to store data in a variety of domains such as business, healthcare, passport verification, and so on [2]. It used SHA-256 hashing cryptography algorithms for security and can be used to store cryptocurrency transaction logs. Blockchain has significant potential in internet-of-things (IoT) applications because it strengthens device security, and data obscurity, while also improving maintainability.

2.1.1 Technologies supporting the blockchain

The main technologies that support the blockchains include, decentralized consensus and storage, public-key cryptography and asymmetric encryption, smart-contracts, and smart contracting languages.

Decentralized consensus: The process of updating records in a blockchain network to ensure that new information is accurate and consistent, is known as blockchain consensus [157]. A blockchain consensus system ensures that only correct data, validated by collaborating parties, is added to the network for an IOC using decentralized collaboration. As a result of blockchain consensus, transparency in IOC processes is improved. Proof-of-work (PoW) is the primary consensus mechanism used by the Bitcoin and Ethereum cryptocurrencies. Bitcoin is the fastest growing blockchain network, and Ethereum is the largest blockchain network for executing smart-contracts [11]. In the PoW consensus mechanism, a complex mathematical puzzle is presented, and the first member of the network to solve it is chosen to add the next record to a blockchain ledger. Because of the scalability and resource consumption issues with PoW, Jain et al. investigated a proof-of-stake (PoS) consensus method as a better solution. Participants in a PoS network are selected to add the next record to the ledger based on the amount of stake they have deposited [82].

Public-key cryptography (PKC): The PKC is a system that uses a public-and-private key pair to uniquely identify participants in a decentralized network. The public key is used to identify IOC participants, while the private key is used to sign transactions. As a result, PKC provides a tamper-proof source verification system for any activity in an IOC. The asymmetric encryption provided by the PKC can be used to implement access control on IOC processes, thus ensuring that specific IOC functions can only be performed by certain network parties. As a result, PKC significantly addresses the security issues currently encountered in traditional blockchain systems for executing SCs within IOCs that use decentralized collaboration.

Decentralized storage: A blockchain network's records are replicated in all participating nodes [136]. Decentralized databases can be used to extend existing blockchain storage by providing additional repositories for blockchain systems [37], thus preserving digital assets exchanged in IOC-executed blockchain networks. As a result, for an IOC that employs decentralized collaboration concepts, decentralized storage ensures that data resulting from the execution of IOC functions are accessible in real-time to all participants. As a result, interoperability issues in current IOC systems are eliminated by real-time data access.

Smart contracts: Smart contracts are blockchain-based applications that allow businesses to be more efficient by automating business processes [98]. A smart contract is a computerized transaction that enforces agreement rules automatically without the use of intermediaries [160]. Collaboration processes between organizations can be reconstructed into smart-contract workflows and executed on blockchain networks. Smart contracts can be coded with business logic and rules, thereby ensuring that SCs in IOCs are executed securely without requiring centralized authority. Smart contracts that are blockchain enabled do not interact with external data; they rely solely on data provided by the blockchain system to carry out business logic. Decentralized oracles are external data gathering components in a blockchain that allow SCs to receive real-world data inputs without relying on centralized parties [65]. This ensures that SCs use trusted data inputs when executing business logic in IOCs.

Smart contract language: Smart contracts are written in a blockchain programming language, such as Solidity, with intermediate languages like Simplicity [165] and Scilla [151] being used for program analysis and verification. The latter provide significant assurances by relying on type-soundness. These languages allow smart contract code to run on low-level virtual machines (VM) (Ethereum VM, for example). Rootstock [4], Telegram Open Network (TON) [5], and Bitcoin [84] are just a few of the blockchains that, like Ethereum, have designed and implemented, their own virtual machines. Rootstock, for example, added the Rootstock Virtual Machine (RVM) to Bitcoin [84], whereas TON introduced the TON VM, or TVM, for developing, maintaining, and configuring SCs [50].

2.1.2 Legal implications of smart contracts

For contracts to be legally binding, the parties to the contract must reach an agreement. According to Governatori et al., [70], the conceptual links between legal commercial contracts, and smart contracts, are that SCs must meet specific requirements in order to be legal contracts. These requirements include: offer and acceptance, consideration, competence, capacity to contract, and so on. According to Savelyev [147], SCs are in accordance with Roman contract law, which he explained by comparing smart contracts to the mechanism of a vending machine. When using a vending machine, an individual places a coin in the slot, which leads to a secure lockbox. The individual then selects a specific product from a predefined list, and if the money deposited in the machine is of the same value as the product selected, the vending machine automatically dispenses the product.

Savelyev also proposed blockchain decentralization as a solution for aligning with government power. The proposed solutions are based on granting state authorities the ability to modify blockchain databases as superusers, while also emphasizing traditional remedies and enforcement practices. Furthermore, Goldenfein et al., [68] argued that the legal status of SCs are dependent on the incorporation of computational transactions into natural contracts, because "natural contracts do not construct an agreement on their own".

[4]Rootstock (RSK) | Home Page

[5]TON | GitHub Page

De Filippi et al., [63] defined a smart contract as “law is code” and proposed abandoning the concept of “code is law.” It is worth noting that when contract law is translated into smart-contract code [62], the semantics of contract law are lost, thereby leaving the legal status of SCs in doubt.

2.2 State of Art of Smart Contract Development

Smart contracts using distributed ledger technology are a new field of study. As a result, little research has been done on SCLs that support the development of legally-binding SCs that are based on the requirements and properties of actual business processes. To support these contractual properties, attempts have been made to raise the level of abstraction from code-centric to model-centric smart contract development. Four of these attempts are described in detail in this chapter; these are: the agent-based approach, the business process-based approach, the state machine approach and the UML approach. One or more modeling languages have been used in these model-driven approaches in order to support the concerns and viewpoints associated with SCs.

2.2.1 Agent-based approach

Frantz et al., [64] developed a modeling approach that used a domain-specific language (DSL) to help transform institutional concepts into machine-readable contractual rules. This approach was based on the concepts discussed in [36]. Crawford & Ostrom proposed a ‘grammar of institutions’ syntax that can be used to identify essential components of any institution and group them into three types of institutional statements: rules, norms, and shared strategies. These institutional statements are useful in agent-based modeling because they serve to guide the actions of agents within organizations [154]. Agent-based models (ABMs) are computational models for simulating the actions and interactions of individual agents, or groups of agents, within a system, for the purpose of understanding agent behavior, and behaviour of the system as a whole. This includes the interactions of the system’s entities as well as specific representations of those entities [153]. The syntax of the grammar of institutions contains five components represented by the acronym ADICO, which stands for: Attributes, Deontic, aIm, Conditions, and Or Else.

Frantz et al., [64] developed a DSL in Scala to convert the ADICO statements into Solidity code. In this way, a business contract written in ADICO syntax could easily be converted into a smart contract using Solidity code. The smart contract, thus generated, would be understood by both IT, and business professionals. However, the code generated through this process would only be a skeleton, and would have to be enhanced by a developer before it could be run in Solidity. Furthermore, no mention is made in this paper of the implementation of such an approach for the creation of complex collaborative business contracts.

2.2.2 Business process-based approach

This approach focuses on organizational business processes, and there have been several initiatives based on this approach published in the literature. To address the lack of trust in collaborative process execution in the blockchain, Weber et al., [169] proposed an automated way to generate smart contracts using a translator. This is achieved by generating SCs from process specifications using Business Process Model and Notation (BPMN). Because the translator is called at design time and the roles of the participants are unknown, the output of this process results in what is known as a factory, or generic, contract. As a result of the transformation’s adherence to workflow patterns, not all BPMN elements are capable of translation.

Taking a similar approach, Tran et al., created Lorikeet; a model-driven engineering tool that generates SCs from BPMN specifications [163]. The modeler user interface is linked to three back-end components: a BPMN translator, a registry generator, and a blockchain trigger. A business process model is fed into the BPMN translator, which generates a smart contract written in Solidity code. The trigger communicates with an Ethereum blockchain node to compile, deploy, and interact with SCs. Unfortunately, Lorikeet does not support all BPMN notation for translation, similar to the translator proposed in a previous study [169].

Caterpillar [132] is a tool available in the market that is an alternative to Lorikeet. It is a free, open-source, tool that supports advanced BPMN control flow elements like subprocesses, multiple instances, and event handling. However, Caterpillar does not support the modeling of business process views, which is essential for any collaborative business contracts.

2.2.3 State machine approach

Several studies, have used the state machine approach to extend model-driven engineering concepts to smart contract development. This researcher considers SCs to be state machines; the contract has an initial state, which changes as transactions are completed. This method is listed as a common pattern in Solidity documentation [40]. Mavridou et al., [106] presented a formal model for modeling SCs and created the FSolidM tool; a code generator for creating Ethereum smart contracts, thus enabling smart contract development with minimal manual coding. In addition, Mavridou et al., demonstrated the FSolidM tool in [107]. Although the transformation from finite state machine (FSM) to Solidity is semi-automated to ensure good code quality, several other properties cannot be modeled without the use of additional plugins. Mavridou et al., [107] did not go into de-tail about FSM; they expanded on previous work by developing the 'VeriSolid' framework, which focused on the security aspect of smart contract design [109]. The VeriSolid framework enabled developers to perform high-level verification of smart contracts, thus allowing for correct-by-design contract development. There is a tool, known as the 'Yakindu Statechart Tool' that has been designed to generate SC code from a finite state model. The tool allows for graphical editing of statecharts, and the generation of code in blockchain languages including Solidity, Vyper and Yul [118]. This tool is, however, in its initial development phases, and no details are avalable on how it can be implemented.

2.2.4 UML approach

Syahputra et al. [158] generated SCs for two different blockchains using UML and OCL. The smart contract code was generated with the help of an existing code generator called 'Acceleo' (Model to Text) [78]. To accomplish their goal, Kruijff et al., employed the commitment-based ontology perspective [41], which segregated ontology into three levels: essential, infological, and datalogical. However, this study [41] lacked detail because it only demonstrated the models and did not discuss platform-specific models or generated code. It is also not clear whether the author's framework integrated the two tools into a single system or whether developers would need to use other tools to achieve their goal.

2.3 Running Cases

Two running cases have been used in this thesis to introduce and discuss the concepts of business contractual collaboration. The first running case is from the automotive sector presented in Publication (II), in which it is assumed that a CarMan manufactures automobiles and outsources a significant portion of the supply chain to collaborating parties who act as service providers or sellers.

The second running case is from the food supply chain, and addresses the issue of process views and their matching relationships, as presented in Publication (IV). Process views are an important consideration when establishing inter-organizational smart contract collaboration. Companies are in full control of the development of process views, which are abstractions of actual business process. In this way, sensitive or insignificant aspects of the supplier business process can be hidden behind a process view.

Both running cases are used to demonstrate the generality and applicability of proposed or developed artefacts. In this thesis, the first running case, the automotive manufacturer, is used to evaluate the proposed the SLCML ontology for configuring essential contractual properties that address conflicts in rights and obligations between contractual parties in an automotive supply chain. The second running case is used to assess the SLCML's syntactical correctness, semantic usefulness, and effectiveness. Using laboratory experiments, It can be assumed that ontology is tested in both running cases because ontology is the input for developing SLCML language.

2.3.1 Case 1: Decentralized system of automobile collaborative supply chain.

Blockchain has many different applications in the automotive industry [176], and the monitoring and control of components or parts in the automotive supply chain is a significant use-case for blockchain [28]. The first running case, which was established in [120] and published in [54], discusses the P2P DAO-collaboration model depicted in Figure 3. Figure 3 illustrates a service consumer's in-house business processes in the form of a business network model (BNM) [143]. The BNM is essentially a blueprint for inter-organizational collaboration that includes a legally binding template contract that matches service types to organizational roles.

'Business network model selection' is a platform for developing service types that can be used in tandem with cloud-based collaborative business-processes-as-a-service (BPaaS-HUB) [127] to support in-house processes. Service offerings from multiple supplier organizations are stored on a BPaaS-HUB, allowing customers to locate potential collaborating partners and learn more about their offerings. The role of a BPaaS-HUB is to match customers with service suppliers, and vice versa. An eSourcing Markup Language (eSML) developed by Norta et al., [128] has been used to specify BNM requirements.

As service offers are received, they are validated against the service types defined in the BNM. For those offers that match requirements, a prototype contract is created for the purpose of negotiation [122]. The negotiation phase may result either in a consensual agreement (acceptance of offer), a counter offer, or a disagreement (Rejection of offer). A prototype smart contract is sent to the all parties who respond to a service offer request, and who meet the criteria. Parties who receive the prototype contract can vote on one of three negotiation options (agreement, counter offer, or disagreement). Successful negotiations result in the creation of a smart contract that satisfies all parties.

As discussed in [122], the service type, service offer, and service role are negotiated in two stages. Stage 1 involves the extraction of proto-contracts, while Stage 2 involves the establishment of SCs. According to [92], agent-based negotiation is rapidly gaining acceptance because it allows for semi-automated and fully automated negotiation.

Figure 3 approximates a BNM representation of a DAO. The model is divided into six zones; the top zone is the legacy technology layer, the next zone represents the conceptual layer for the service consumer (the buyer of services). The conceptual layer represents the workflow instances that the service consumer is prepared to make visible to the external world. It should be noted that these are not the actual workflow processes, but

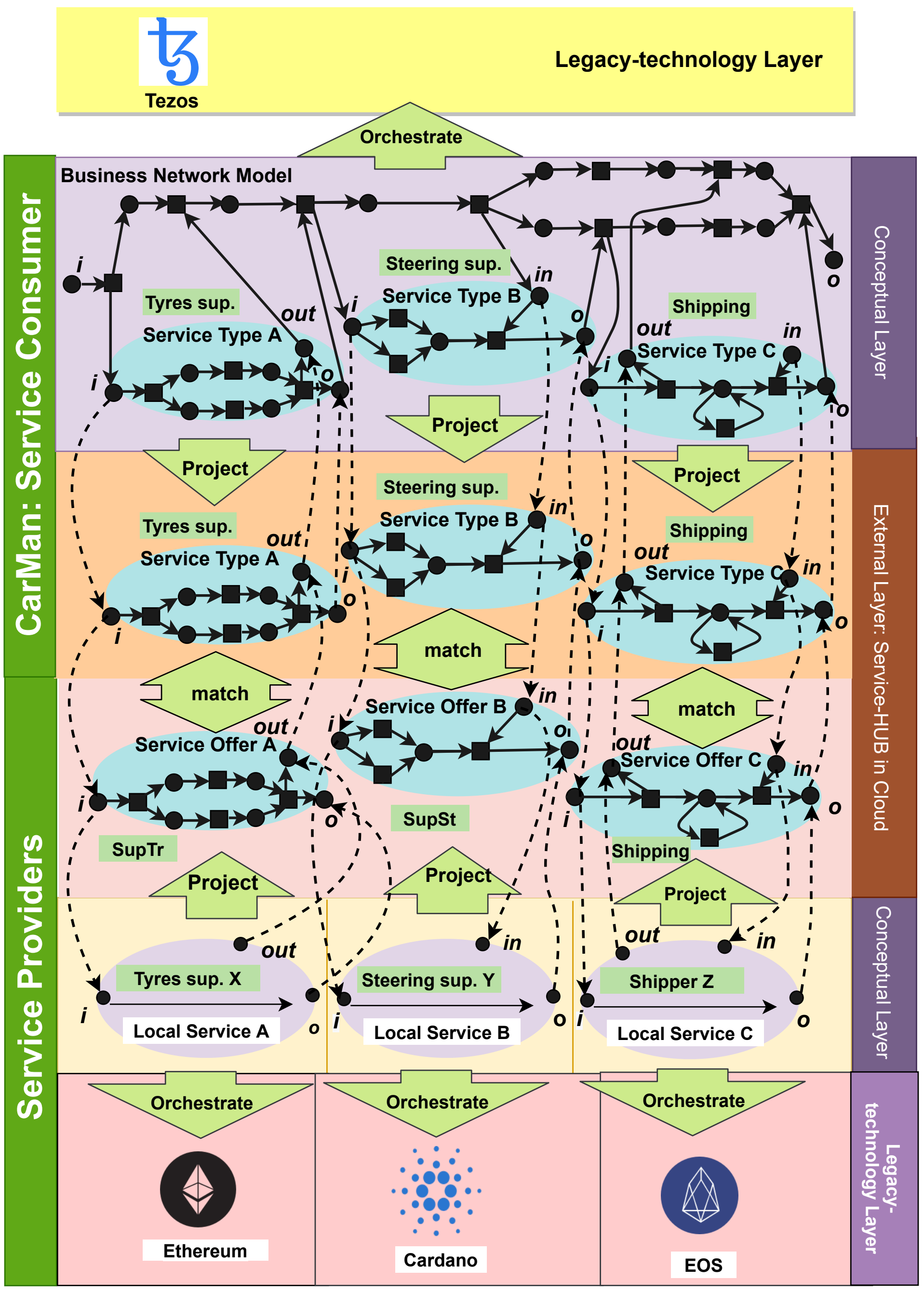


Figure 3: DAO-collaborative automotive supply-chain adopted from Publication (II).

are representations of those processes that the service consumer needs to make visible to potential service providers. In the context of Figure 3, these workflow representations are projected by the service consumer into the business-process-as-a-service hub in the cloud; as represented by the external layer.

By the same token, the second-to-bottom layer of the model represents the service providers, and they too, only make visible those parts of their internal workflows that are necessary to satisfy the needs of potential customers. These representations of internal workflows are projected into the external environment by way of the BPaas-Hub; this is what the service providers (suppliers) are offering in terms of products or services. A matching process takes place in the middle (the external cloud-based service hub layer), where service requests of the service consumer (manufacturer or buyer) are matched with the service offerings of the service providers (suppliers). A successful match results in a negotiation, and a successful negotiation results in consensus and the creation of a smart contract.

In Figure 3, the CarMan is a car manufacturer (classified for the purpose of this thesis as a service consumer), who manufactures and assembles cars from components and parts provided by a network of suppliers. There are three service providers: Supplier A (SupTr) is a manufacturer of car tyres, supplier B (SupSt) manufactures steering wheels, and Supplier C (Shipping) delivers components and parts to CarMan, and delivers assembled cars from CarMan. All four entities are involved in inter-organizational collaboration in a DAO for the purpose of manufacturing automobiles. The CarMan, is a focal manufacturing operation which controls a variety of internal processes, such as process control, information exchange, and resource planning for the manufacture and sale of automobiles. However, because CarMan is unable to manufacture all of the components and parts internally, it sources those components and parts it needs from external suppliers and service providers.

Two technology layers are illustrated in Figure 3; at the top there is the legacy technology layer, where the service providers' internal processes interface with their legacy information technology systems. The second technology layer; the smart-contract blockchain technology layer, is shown at the bottom of the diagram. This illustrates how various smart-contract blockchan solutions such as Ethereum, Cardano, the Electro-Optical System (EOS) and Tezos among others, may be mapped to the respective internal legacy-technology layers of the parties collaborating in the DAO. The dashed monitorability- and conjoinment arcs [126] show how the proposed conceptual business processes are linked to the external layers and how they can be realized technically with the lightning network [91]. Lightning is a decentralized blockchain-based network using SC functionality to enable instant payments for high-volume transactions to participants within a DAO, without entrusting funds to a third party.

For the purpose of this thesis, only cross-entity market exchanges have been considered. CarMan generates demand for car parts by using blockchain SCs to specify quantities, prices, delivery dates, and other criteria. CarMan's demand for products and services is conveyed to service providers through public blockchain platforms, and they respond with bids that include the terms and conditions relating to the provision of the car parts. A smart contract has rules that can not be changed or opened before the deadline [30] to protect the integrity of bid prices and private information provided by service providers. For additional security, the bids submitted by public blockchain participants can be encrypted before they are submitted. These bids can then be decrypted on receipt by using a decryption key held by the software agent receiving the bids. Because the details of the bids are stored on secure blockchains, CarMan is able to select suppliers either manually or automatically if specified criteria have been met. In addition specific provisions related to supply chain collaboration can be coded in the respective legacy-technology layers. These provisions would automatically trigger specific pre-defined actions if certain events occur. For example, if the tyre manufacturer failed to deliver car tyres to CarMan on time, the SC could be set to penalize the supplier prior to delivery. In a traditional supply chain, cooperating entities have little or no say over who is to blame for bottlenecks. To achieve this oversight, SCs and blockchain technology are used, allowing collaborative parties to monitor and track the status of products and transactions.

Although SCs and blockchain technology can be used to improve and automate the process, thus allowing parties to monitor and track the status of their products and transactions, there are valid business concerns that can arise from using evolving technology. For example, a SC may be programmed to automatically release cryptocurrency on the delivery of car tyres based on receipt of tyres into a warehouse, even though the delivered product does not meet CarMan's specification. The legal implications are that payment has automatically been made for defective goods.

SupTr has delivered defective tyres to CarMan, who has automatically paid for them on receipt. CarMan would have a claim on SupTr, who would then have two options: 1) replace the defective tyres at their cost, or 2) refund the money. The obligation to fulfill that compensation must be imposed on the SupTr. Another possibility is that the steering wheel supplier sells steering wheels to CarMan, but the product is not delivered by CarMan's deadline due to a conflict with the Shipper. Both of these scenarios have practical implications that SC-based blockchains must be able to cater for.

Traditionally, international trade issues such as these, have been resolved through the use of letters of credit, in which the buyer receives assurance that the cargo price will not be paid unless the seller has proved that he has met all the the obligations assigned to him under the underlying contract of sale. When the terms of the contract have been met, the seller receives his money, and the bank is compensated for acting as an intermediary in this transaction by charging the buyer a fee [89]. Nonetheless, as a sluggish and outmoded paper-based payment method requiring both parties to exchange and verify official-and-legal documents, this payment method faces numerous challenges. Furthermore, rather than relying on the underlying condition of the goods, this payment method is entirely dependent on documents to initiate payment [3]. Because of the need for "physical documentation exchanges," as well as the transfer bill-of-lading and complex communications between many different parties; paper-based letters of credit are time-consuming, and prone to errors. These disadvantages can be overcome by implementing blockchain-based SCs to reduce the time required for credit transactions through enabling electronic payment of bills-of-lading. In addition, the need for paper-based documents could be eliminated, and the interaction between all parties would be facilitated through a single, private network. However, contractual semantics in existing SCL are insufficient to implement blockchain-enabled letters of credit.

2.3.2 Case 2: Collaborative dairy supply chain for business processes, process views

The second running case of this thesis as elaborated in [15], and published in [53], describes numerous problems that can arise when a service provider (supplier) hides process details from the service consumer (customer) in the dairy supply chain. Blockchain technology may benefit the food supply chain, such as the pork supply chain [29], the fish supply chain [79], and the dairy supply chain. The tracking and monitoring of product safety and regulatory compliance throughout the food supply chain is a significant use-case for blockchain [26]. Many stakeholders, including farmers, bulk milk distributors, manufacturers,wholesalers and retailers, as shown in Figure 4, are responsible for managing the supply-chain operation from the start, when a cow on a farm produces raw milk, to the finished product, when a consumer buys baby-milk powder. Internal traceability refers to the traceability of one of the actors internal processes, whereas chain traceability refers to the traceability of the entire supply chain [115]. To retrieve and provide information to the Food Safety Information System, actors in the dairy supply chain can employ IoT devices and location-based food safety information systems (FSIS) technology. The latter contains a wide range of data that food supply-chain actors require in order to achieve transparency and quality assurance. According to [10], FSIS is run by unspecified centralized or decentralized information. Each actor is expected be responsible for food safety in their own operations. The Food Safety and Quality Assurance System (FSQAS) establishes the quality and safety standards to which all stakeholders in the supply chain must adhere. The FSIS monitors traceability data to ensure that rules are followed.

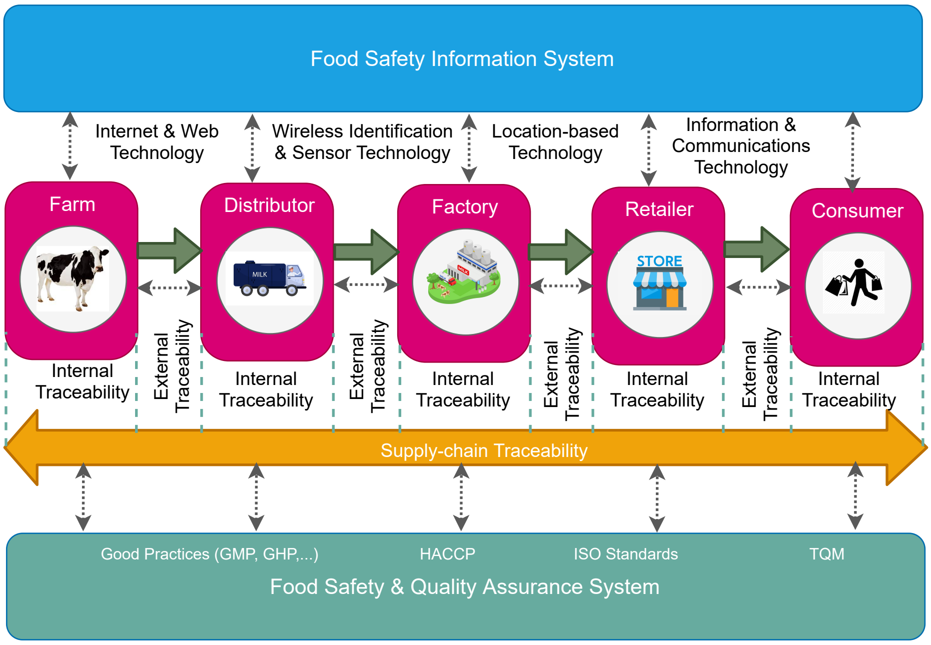


Figure 4: Use case of dairy food supply chain adopted from Publication (IV).

This thesis focuses on cross-organizational collaboration within the dairy supply chain. Farmers keep detailed records of their farm's location, breed of cow, vaccinations, treatments, and any special regimens that might be required. RFID devices or any other sensor network incorporating blockchain technology can be used to monitor the health and movement of animals. Using advanced machines, data on animal movement can also be recorded and stored on blockchains. The bulk milk distribution company is informed through blockchain platforms when the milk is ready for collection. Temperature control during transit is critical for preventing milk spoilage, and sensors are used to achieve this. GPS technology is often used to monitor vehicles in real time. When the milk is dispatched to the factory, key information is updated on the blockchain network. These data include information such as the location of the unit, the number of deliveries at a specific lot, and so forth. The factory processes the milk and manufactures baby-milk powder; in addition is also provides consumers with factual data about food items such as: nutritional information, ingredients, expiry date, instructions for use, and other helpful or legally-mandated information.

According to [27], SCs are required for food supply-chain operations to improve collaboration among parties. The supply-chain operation procedures in the food-safety and quality-assurance system are designed to trigger important events. If, for example, the bulk milk distributor fails to deliver milk to the factory within a specified time, or at a specified quality, they may end up paying a penalty. If this scenario is considered to be important it should be coded into the SC. In a traditional supply chain, the parties who work collaboratively, often have little control over any organization that could cause a bottleneck in the system. However, in a SC-driven blockchain, each business could monitor and track the status of products and transactions, thus providing critical oversight of the whole process. Any bottleneck would immediately become visible, and the erring party would be identified. Despite these advantages, there are significant business and legal concerns relating to the fact that blockchain technology is still in early stages of development, and SCLs are not mature enough to deal with practical realities. Assume, for example, a farmer is managing a workflow process on behalf of the milk processing factory. Even though the farmer is supplying bulk milk to the factory under a supply contract, he may not wish to reveal the details of all the workflow processes he employs on his farm, because they are of no concern to the factory. The factory, on the other hand, may believe that because they are the farmer's customer, they have a right to have insight into what happens on the farm. The farmer only discloses those aspects of the process he is willing to make public, and that are of interest to potential customer organizations. Most customer organizations, on the other hand, would like to incorporate a white-box view of the outsourced processes into their own processes. This would provide them with more information about the structure and progress of the process that another company is carrying out. However, in reality, the customer organization, does not need to know the specifics of the upstream workflow process and only requires a broad understanding. To accommodate issues like these, SCs must include clauses that clearly define the rules for when confidential workflow information should be withheld or disclosed to third parties.

3 LEGALLY BINDING SMART-CONTRACT LANGUAGE DEVELOPMENT FRAMEWORK

This Chapter focuses on RQ1: How can a novel framework be devised for designing smart contract languages that are semantically rich and which support the drafting of formally verifiable smart contracts, for use in DAO collaboration? In Publication (I), a legally binding SCL development framework was proposed, and the contractual business concepts and properties that enable SCLs to facilitate the drafting of SCs supported by law, was discussed. The analyses, and the results produced as a result of that analysis, have been used as the contents for this chapter.

3.1 Introduction

In this chapter, a comprehensive meta-study of existing smart contract languages is conducted by way of a systematic literature review (SLR). The focus of this meta-study is on the coding of smart contracts for distributed autonomous organizations.

This researcher follows Kitchenham's guidelines for conducting a software engineering SLR [85] and conducted the study using a structured five-step process. The study begins by identifying relevant keywords (and their synonyms) commonly used in scientific literature that are relevant to the specific research questions. The 616 articles published as white, and grey, literature are identified by conducting a thorough search of various academic databases (including Web of Science, Scopus, arXiv, and Google Scholar), and technology-related online publishing platforms including GitHub [Publication (I)]. Using a predefined set of inclusion criteria (IC1-IC6) [Publication (I)], and exclusion criteria (EC1-EC6) [Publication (I)], the titles, keywords, and abstracts of these articles are examined to extract relevant articles that propose (or discuss) novel SCLs. The selected 130 *primary studies* [Publication (I)-supplementary[6]] are then thoroughly studied and scored using a two-stage quality-assessment process (QC1.1-QC1.6 and QC2.1-QC2.4) [Publication (I)]. The 73 articles that explicitly answer the research questions *SRQ1.1* and *SRQ1.2*, are identified and labeled as *selected studies*, and *supporting studies*. Following that, the 45 SCLs proposed in the identified *selected studies*, are examined and categorized based on their attributes and applications. Available properties are identified and new ones that are critical when developing legally binding SCLs are proposed by examining the remaining 28 *supporting studies* [see Publication (I)- Table 9 and 11]. Finally, to determine whether the identified SCLs have the required suitability and expressiveness properties for developing DAO SCs, the selected-supporting studies and their references are thoroughly examined. Based on the findings of the SLR, this researcher concludes that most cutting-edge SCLs only partially support business' contractual processes. A novel framework is developed for designing semantically rich SCLs that support the drafting of formally verifiable SCs aimed at DAO collaboration.

3.2 Existing SCLs

This researcher examined and summarized the characteristics of the aforementioned SCLs, including the *name* SCL, the *blockchain* platform for which it was designed, its *type-system*, *paradigm*, *focus*, and *purpose*. These properties were chosen because they are critical for understanding the differences between current SCLs, and for identifying critical properties that can aid in the creation of legally binding SCs.

The attributes: *focus* and *purpose* describe the motivation for SCL development; for ex-

[6]Publication 1 | Supplementary material

ample, the specific domain/activity for which the SCL was created, as well as whether the proposed SCLs were detailed enough to be checked for semantic correctness, formal verification, or both. The *paradigm* attribute, determines the programming paradigm or execution model of SCLs (process-flow or data-flow). Finally, the SCL *type-system*, specifies the rules that apply to the data types of the programming language. These characteristics are especially significant because they reveal critical information about the suitability and expressiveness of SCLs. In particular, *focus* and *purpose* provide insights about semantic suitability; *focus* and *paradigm* provide insights about workflow suitability; and *purpose* and *type-system* provide insights into expressiveness.

- *SRQ1.1*: What blockchain-based SCLs already exist in scientific and non-scientific literature?

Because the number of selected SCLs are too large to be described individually, they have been grouped into five categories based on their foci: domain-specific SCLs, formally verifiable SCLs, easy-to-use SCLs, legally enforceable SCLs, and business process SCLs.

Domain-specific SCLs are those SCLs that have been designed with a specific domain. *Formal verifiable* SCLs include languages that are designed with priority to runtime safety of smart-contract code. *Easy-to-use* SCLs are simply languages that are human understandable, or lay-person friendly. *Legally-enforceable* SCLs are intended to, or actually do, result in legally-binding contracts. Finally, *business-process* SCLs are languages that are specifically designed to automate business processes.

Table 4: Details of SCLs presented in selected studies [Publication (I)].

SCL	Ref.	Blockchain	Type-System	Paradigm	Purpose*	Focus$^+$
ADICO	[64]	Ethereum	Dynamic	Declarative	SPEC	Legal-Contracts
Babbage	[32]	Ethereum	Type-Safety	Symbolic	SPEC	Human Undr.
BALZaC	[9]	Bitcoin	Dynamic	Imperative	SPEC	Verification
Bamboo	[179]	Ethereum	Type-Safety	Imperative	IMPL	Formal Verf.
BCRL	[6]	Hyperledger	Dynamic	Declarative	IMPL	Business Process
BitML	[12]	Bitcoin	Dynamic	Declarative	IMPL	Security
Commit-Rule ML	[44]	-	Static	Declarative	SPEC	Legal-Contracts
DAML	[46]	Hyperledger	Dynamic	Declarative	IMPL	Business Process
DSL4SC	[161]	Hyperledger	Dynamic	Declarative	SPEC	Natural Language
ErgoScript	[45]	*Independent*	Type-Safety	Declarative	SPEC	Legal-Contracts
eSML	[128]	-	Dynamic	Declarative	SPEC	Business Process
Fi	[4]	Tezos	Type-Safety	Imperative	IMPL	Verification

Fift	[50]	TON	Dynamic	Imperative	IMPL	Domain Specific
Findel	[17]	*Independent*	Dynamic	Declarative	SPEC	Financial Contracts
Flint	[148]	Ethereum	Static	Imperative	IMPL	Security
Formality	[103]	Ethereum	Static	Declarative	IMPL	Efficiency
FSolidM	[108]	*Independent*	Dynamic	Declarative	SPEC	Security
F-Sol	[141]	Ethereum	Static	Functional	IMPL	Verification
Idris	[133]	Ethereum	Dependent	Declarative	IMPL	Security
IELE	[83]	IELE	Static	Imperative	IMPL	Verification
Ivy	[142]	Bitcoin	Static	Declarative	IMPL	Domain Specific
Liquidity	[130]	Tezos	Dynamic	Functional	IMPL	Formal Verf.
LLL	[58]	Ethereum	Dynamic	Declarative	IMPL	User Frnd.
Lolisa	[177]	Ethereum	Static	Imperative	SPEC	Formal Verf.
Marlowe	[150]	Cardano	Dynamic	Declarative	SPEC	Financial Contracts
Michelson	[162]	Tezos	Monomorphic	Low-Level	IMPL	Domain Specific
Move	[18]	Libra	Static	Imperative	IMPL	Verification
Mutan	[172]	Ethereum	Dynamic	Imperative	IMPL	Formal Verf.
Obsidian	[34]	Hyperledger	Static	Imperative	IMPL	Security
Pact	[137]	Kadena	Dynamic	Declarative	IMPL	Security
Plutus	[25]	Cardano	Dynamic	Declarative	IMPL	Financial Contracts
Pyramid	[23]	Ethereum	Strongly Typed	Imperative	IMPL	Safety
QSCL	[38]	Qtum	Static	Imperative	IMPL	Business Process
Reaction-Rule ML	[43]	-	Static	Declarative	SPEC	Legal Contracts
Rholang	[112]	Rchain	Dynamic	Declarative	IMPL	Domain Specific
RIDE	[14]	Waves	Static	Declarative	IMPL	User Frnd.
Scilla	[151]	Zilliqa	Static	Functional	IMPL	Security
Script	[117]	Bitcoin	Static	Imperative	IMPL	Crypto.
Simplicity	[164]	Bitcoin	Dynamic	Functional	IMPL	Security
SmaCoNat	[140]	-	Dynamic	Imperative	SPEC	Natural Language
Solidity	[61]	Ethereum	Static	Imperative	IMPL	Domain Specific

Sophia	[174]	Aeternity	Strongly Typed	Imperative	IMPL	Domain Specific
SPESC	[76]	-	Dynamic	Declarative	SPEC	Legal-Contracts
Typecoin	[35]	Bitcoin	Type-Safety	Symbolic	IMPL	Crypto.
Vyper	[24]	Ethereum	Dynamic	Imperative	IMPL	Security

* *SPEC*: Specification. *IMPL*: Implementation
+ *Verf.* [Verifiable] | *Frnd.* [Friendliness] | *Undr.* [Understandable] | *Crypto.* [Cryptocurrency]

3.2.1 Domain-specific SCLs

Solidity is a DSLfor creating complex SCs for digital assets such as voting, crowdfunding, and so on [151]. It is a statically-typed language with multiple inheritance and complex user-defined data types. Solidity contracts are finite state machines that prevent any transaction calls to other contracts from being made while in a state transition. A transaction that modifies the state of a contract is either successful or unsuccessful. According to [113], because Solidity has a number of bugs and vulnerabilities, such as *re-entrancy*, and *delegatecall*, several tools and frameworks, such as Oyente, Mythril, Securify, and others, have been developed to verify, and analyze, Solidity code. In order to overcome Solidity bugs, Ethereum introduced the Vyper language [113], which focuses on security, audibility, and simplicity. Vyper's goal is to make it more difficult for developers to intentionally write malicious code. Vyper also makes use of the built-in libraries of integer (*overflow/underflow*) to prevent unintentional security flaws in the code. Solidity does not perform any checks on transaction return values due to caller contract exceptions. To handle such exceptions, Vyper implements the *send()*, and *raw-call()* functions; if these fail, the entire transaction is reverted. Furthermore, when a contract calls an external contract, Vyper provides functions to prevent re-entry vulnerability. If necessary state changes are not performed prior to calling an external contract, the contract is especially vulnerable to re-entrance attacks. Vyper implements the *nonreentrant decorator* [7] that places a lock on the current function, and all functions with the same key value. Idris, another SCL, was created to secure smart contract code by using *dependent types* [72]. Idris permits types that are dependent on values, implying that types are first-class language constructs that can be manipulated in the same way as any other value.

Flint is a contractually-specific and statically-typed SCL that compiles into EVM bytecode using the intermediate language YUL [133] [8]. YUL was designed by the Solidity team to work with a variety of EVM backends, including EVM 1.0, and EVM 1.5, as well as a variety of front-end languages. Flint's goal is to write smart-contract code that is inherently safe and predictable, as a result it does not have to be analyzed it after it is written, unlike Solidity. To prevent unauthorized calls to contract-sensitive functions, Flint employs a *caller capability block*, which declares the right to call Ethereum user accounts. Flint ensures contract state consistency and provides safer atomic transactions. Formality SCL is more time-efficient in contract execution than Solidity because its core is built as an affine lambda calculus, which allows it to be garbage-collection free [103]. It is statically-typed and has the appearance of a Python-style programming language with formal proofs. Pyramid Scheme is a functional and mandatory SCL that encourages the separation of state-changing and static functions. The Pyramid Scheme also has an eye on

[7]Vyper | Documentation on Structure of a Contract
[8]Yul | GitHub Page

the EVM [23]. The functions are designed to be atomic, and executed completely, to ensure consistency in smart-contract state changes. In Pyramid Scheme, pure functions are also used to indicate that no effect on the global or local state exists. Findel is a declarative and domain-specific SLC designed for securely handling financial agreements in the EVM. Findel is a formal language that distinguishes between contract description and contract execution, limiting the formalization of unambiguous contractual clauses [17].

Michelson is a low-level stack-based language created by Tezos developers [162]. Michelson's instructions are executed with an unrestricted stack-length, thus ensuring that the code is executed securely. When it comes to SCs, Michelson differs from Solidity in that it aims to write a piece of business logic. Ethereum contracts are written to implement concepts such as multisig wallets, vesting, distribution rules, and so on, and Michelson is not for writing arbitrary programs, it is targeted to specific applications. Plutus core is a blockchain transaction-validation system, that is implied to be a compilation target, as expressed in the language's design. While writing large Plutus Core programs by hand is difficult, formalizing the language with a proof assistant is relatively simple [25]. Plutus-core is used to validate on-chain transactions. The validation process is beyond the scope of this thesis, and the reader is referred to [80] for more information. Rchain implements Rholang [112], a contractual, and concurrent-oriented programming language that focuses on concurrent data, and process, flow to support contractual behavior. Rholang is a process-oriented language, which means that all communication is done through message passing. It has a behavioral type system that allows participants to explore contractual obligations and guarantees in an automated manner, prior to entering into a contractual agreement.

Marlowe is a DSL used to carry out financial transactions [150]. It is implemented as an algebraic type in Haskell programming on UTxO, or account-based blockchains. Pact is a declarative programming language that helps programmers write less problematic code by using a lisp syntax and Haskell-like types [137]. Pact's goal is to enforce business rules that prevent system records on the Kadena blockchain from being updated. Sophia is a blockchain-specific, strictly-typed programming language with a constrained mutable state, and ML-comparable programming capabilities. Sophia is designed specifically for the private Aeternity blockchain, and aims to provide blockchain-specific primitives, constructs, and types [174]. Fift is a stack-based programming language used on the TON blockchain to create, manage, and debug SCs. It is a dynamic-type language for interactive experimentation, debugging, and building basic scripts that uses the stack to store the value of multiple types other than integers [50]. Ivy is a predicate language for creating ChainVM SCs that are intended to be educational [142]. Bitcoin makes use of the Script language, which is a list of instructions recorded with each transaction that describe how they can be accessed by the next person who wants to spend the Bitcoins being transferred [117]. To carry rational propositions, the Typecoin language supports a rational commitment process built on top of Bitcoin. The underlying idea behind Typecoin is that transactions carry logical proposals rather than coins [35]. Each Bitcoin transaction can be converted into a Typecoin transaction, with inputs and outputs transformed into propositions, and logic allowing inputs to be split or merged.

3.2.2 Formally verifiable SCLs

A smart contract, like a traditional contract, can explicitly include rights and obligations semantics. Unfortunately, due to a lack of common understanding between programmers and legal experts, SCs currently contain numerous legal loopholes, which unlike common programming bugs, are not easily discovered. Several static, and dynamic instruments, such as Mythril, Oyente [16], and others, have been developed, but have yet to be proven to secure SCs. As a result, a language's formal verifiability is critical for ensuring the correctness, and run-time safety of smart-contract code. Lolisa [177] is the first mechanized and validated formal syntax and semantics developed for Solidity. Lolisa not only supports Solidity syntax such as mapping, modifiers, and so on, but also supports conventional programming characteristics such as multiple return types and structures. In addition, Lolisa uses a more robust static-type system than Solidity to improve type safety. Bamboo [179] is a formally verified SCL that makes transactions explicit in order to overcome Ethereum contract's reentrancy behavior. Bamboo's programming language supports reasoning as state machines on SCs. Developers define which functions can be called in each state, and the language provides constructs for explicitly specifying state changes.

Furthermore, Ethereum uses Mutan to support the dynamic feature of higher-level languages like C or C++ [172]. The goal of Move SCL is to encode the owners of digital assets, and the business logic that goes with them [18]. As a result, Move provides flexible, safe, and verifiable governance rules for the Libra blockchain [9]. The ability to describe custom resource types with semantic-inspired linear logic [67], is the main feature of Move; a resource can never be replicated or tacitly discarded, only relocated between program-storage locations. Furthermore, Move adds code flexibility by including *transaction scripts*. A transaction script is a one-time function that invokes multiple modules published in blockchain procedures, thus allowing for customizable transactions. Move implements *bytecode verifier*, which checks the Move bytecode on-chain, to fulfill critical security features such as memory safety, type safety, and resource safety. FsolidM [108] is a visual programming framework that is used to define contracts as finite state machines (FSMs). FsolidM includes a code generator for specifying FSMs, which is especially useful when creating Ethereum contracts. Furthermore, FsolidM provides a set of plugins that can be used to improve the security and functionality of FSMs. Plugins are intended to address common design patterns of security vulnerabilities identified in previous work [8, 97]. Obsidian is a state-oriented, static-type SCL that allows the developer to declare and transition states explicitly. Furthermore, Obsidian is based on core calculus, which employs an *typestate* to detect incorrect state manipulation, and an *lineartypes* to ensure that the program manages resources correctly. Obsidian also supports Hyperledger Fabric, a permisisoned blockchain platform.

Fi is a statically-typed language that is designed to be syntactically similar to Javascript, and Solidity, and directly compiles to Tezos blockchain Michelson code. The latter offers a more familiar coding environment that is more akin to an object-oriented programming language. Scilla is an intermediate language developed by *Zilliqa blockchain* [10], which is used as a translation target for high-level languages for program analysis, and verification. Scilla strives for expressivity, and tractability, so that contract behavior can be formally reasoned. Scilla's design principle is based on the separation of computation, and communication, which means that computing the value of a function is implemented independently. Scilla is able to alter a balance without involving any other parties. If involvement is required- for example, transferring control to other parties- a transition would be completed by sending and receiving messages. Existing languages have a distinct specification and implementation, and if the performance differs, it is impossible to execute test cases against the specification. Using the K-framework, IELE aims to bridge the gap between specification and implementation.

BitML is an abbreviation for Bitcoin modeling language, which defines the contract

[9]Libra | Home Page
[10]Zilliqa | Home Page

for regulating Bitcoin transfers using process calculi. Proving the correctness of a smart contract in Bitcoin necessitates proving the computational security of the cryptographic protocol, which increases the programmers' workload. BitML is a symbolic, and computational model for reasoning about Bitcoin security. Participants' will behave in accordance with the semantics of BitML as defined in the symbolic model. A computational model is used to reason about the behavior of the participants' and impose additional restrictions on attackers. Simplicity [164] is a formal semantics-based Haskell functional programming language-based SCL. Its type system supports multiple inheritance, which enables developers to express complex contracts. The primary design goal of Simplicity is to provide statically-computed runtime resource estimations. Liquidity is a high-level, fully-typed language based on the OCaml syntax [130]. The Tezos blockchain created the latter to replace the Michelson language, which is more difficult to read and write due to a lack of stack-based instructions. A compiler generates Michelson code, and a decompiler converts Michelson to Liquidity. Furthermore, Liquidity fully supports the Michelson language by utilizing additional local variables rather than stack manipulations. The functional Solidity [141] language was created to write Ethereum SCs with additional formal methods.

3.2.3 Easy-to-use SCLs

Lisp-Like-Language (LLL) is an Ethereum SCL that translates high-level Solidity code into low-level bytecode and simplifies EVM stack-management. In addition, unlike Solidity, LLL provides a different perspective that does not hide resource perspectives, and al-lows limited resources to be used effectively, as well as facilitating the creation of clean EVM code by directly removing the worst of the coding pain, namely EVM stack-and-jump management. Babbage is a visual programming language that was created to assist non-programmers in understanding complex smart-contract code [32]. By using a vending machine analogy, Babbage hoped to make data-flow transparency a reality.

According to [14], Ride aims to address the shortcomings of existing languages by providing a simple, statically-typed, functional language for dApp development that calculates the amount of gas required for smart-contract execution in advance. DSL4SC is a state machine language for expressing structural, temporal, and constraint properties of state sequences. DSL4SC is a specification language that works with the Hyperledger blockchain. SmaCoNat is a straightforward, human-readable SCL that defines a smart contract in natural-language syntax [140]. SmaCoNat's natural language prepositions are used to define data structure properties. However, SmaCoNat is not a full-featured language; rather, it focuses on creating SCs in natural language with small types and operations.

3.2.4 Legally-enforceable SCLs

This researcher has identfied the SCLs capable of converting legal semantic rules into smart-contract code. The foundations for designing legal SCs are choreography languages such as ADICO [64], ErgoScript [45], CommitRuleML [44], ReactionRuleML [43], and SPESC [76].

To codify laws, the ADICO framework proposed a modeling approach for the semi-automated translation of human-readable contracts into SCs. ADICO programs are built using five rule-based components: attributes, denotic, aim, conditions, and or-else. The *attributes* component represents an actor's characteristics, whereas the *denotic* component represents contract clauses such as rights, obligations, and prohibitions. The *goal, or aim*, of the ADICO program determines the program's outcome. In addition, the condition statement specifies the context in which this statement should be applied. Finally, the *or-else* components describe the prominences associated with a non-conformance.

CommitRuleML, an extension of the 'KR ReactionRuleML', considers a contract to be a commitment-based smart contract. The communication between parties is viewed as a multi-agent system (MAS), with 'MAS commitment' serving as a necessary foundation for the interactions of the organizing parties. CommitRuleML defines the contract using event calculus, and statements like events (on), conditions (if), and actions (do), are defined in executable language. ErgoScript is a real-time scripting language designed to support financial contracts and zero-knowledge proofs. The latter enables the developer to specify the conditions under which currency is spent; for example: who can pay, to whom, and under what conditions, and so on. ErgoScript outperforms Bitcoin Script in terms of power because it lacks a recursive construct that makes estimating run-time difficult. SPESC is a natural language-based specification language for specifying SCs with rights and obligations similar to real-world arrangements, as well as transaction rules. SPESC contains the specification for when specific terms hold, such a description of the parties, a set of terms, and a description of the transaction record. Furthermore, SPESC was intended to facilitate the collaborative development of smart blockchain contracts. However, these foundations are not mapped to high-level programming languages such as Solidity for implementing the semantics of legal SCs into code.

3.2.5 Business process SCLs

Smart contracts on a blockchain enable the completion of business processes. DAML [46], BCRL [6], QSCL [38], and eSML [128] are examples of SCLs that incorporate the semantics of business processes while generating SCs. The DAML is a blockchain functional-programming language with permissions that prioritize business processes over blockchain technology and encryption. The DAML manages rights and obligations by ensuring that contract details are only visible to those who are directly affected by the contract. Furthermore, DAML restricts the number of instructions that can be used to prevent undesirable behavior. The Qtum blockchain develops QSCL that incorporates the semantic-web domain concept and properties. One of the goals of QSCL is the value transfer protocol (VTP) management system, which organizes cross-organizational information logistics and value transfers in accordance with the value proposition. The QSCL specification includes a structure for identifying contracting parties, as well as definitions of resources and data. The eSourcing Markup Language (eSML) is also being developed to facilitate business collaboration by including contractual properties such as party identification, business and legal context, and exchange values. As a result, eSML serves as a choreography language for IOC. A fully functional framework business-level rules language (BCRL) is developed to implement business-core logic in 'Controlled English Language', assisting in the development of a shared understanding between a domain expert and a smart contract developer. A BCRL smart contract can also run on the Hyperfabric ledger as well as an off-chain rules engine, resulting in a more seamless experience for managing general business collaboration solutions.

As discussed above, this researcher has found a total of 45 SCLs: 17 from scientific literature, and 28 from non-scientific literature. Of these, 28 SCLs have been implemented, and the rest have been proposed in academic articles. Figure 5 presents the year in which the SCLs have been implemented or proposed, together with their associated study IDs.

3.3 Suitability and Expressiveness Properties

In this chapter, this researcher identifies the indecomposable properties of SCLs that are important in drafting legally-binding contracts, and classifies them into three categories: 'semantic suitability', 'workflow suitability', and 'expressiveness'.

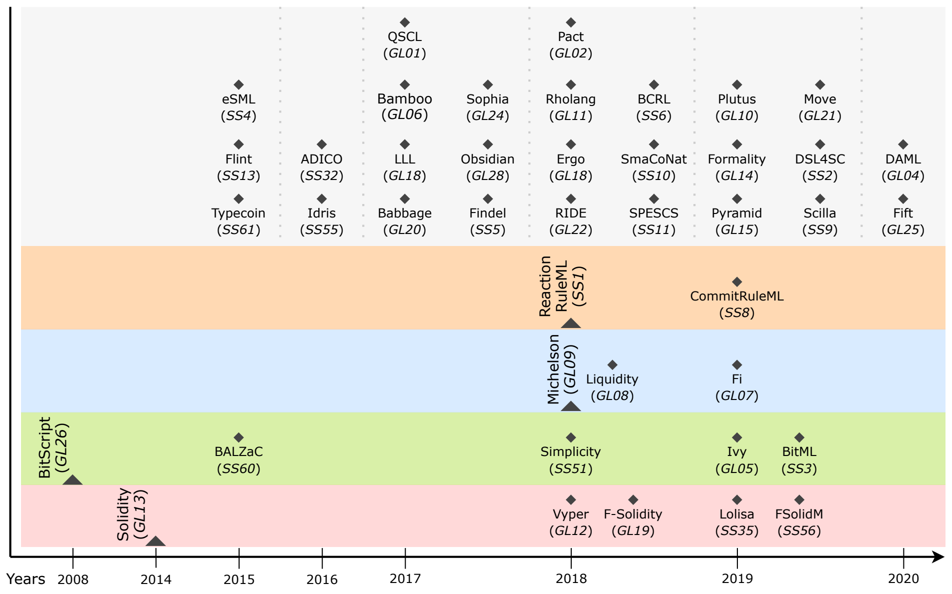


Figure 5: SCLs implementation per year adopted from Publication (I).

Mario et al., [22] provided a unifying theory that aided in the verification of contract compliance in services choreography in their work. Despite the fact that the concept is primarily aimed at the service-oriented computing (SOC) research community, it has significant implications for DAO research as well [121]. Smart contracts in DAOs aim to be ontologically-rich, formally-variable, pieces of code designed to support collaborations between decentralized entities; just as services in SOC aim to be 'self-describing computational elements that support composition of distributed applications' [134]. Drawing parallels between the two: the ontological concepts and properties of SCs, can be equated to the choreography conformance requirements in service design [135, 69]. To put it another way, the choreography conformance requirements could be viewed as semantic, and workflow properties (*suitability*, for example).

'Semantic suitability' properties can be defined as fundamental components, from the perspective of the eContract paradigm [128] and thus, include properties that provide insights into the context; for example, who is participating the in transaction, what they are exchanging, and under what provisions, terms or conditions [70] of a smart contract. 'Workflow suitability' encompasses properties [146] that provide insights into the processes (how the transactions are being carried out), or workflow patterns, from the perspective of the contractual collaboration paradigm [128]. In this thesis, this researcher builds on Norta et al.,'s view of workflow patterns, and introduces business-process modeling (BPM) patterns proposed by Russell et al., [146] as critical properties that can help understand how the state of a contract would change after any given set of interactions.

- *SRQ1.2*: What properties of business-oriented SCLs contribute to suitability and expressiveness?

The goal of this SRQ is to categorize and identify the suitability and expressiveness properties of SCLs. During the analysis of SCLs, it was discovered that they are typically designed around a broad spectrum of foci, which often influence the formal verifiability and semantic correctness of the SCL, as discussed earlier in Section 3.2. As a result, identifying all of the properties that make them legally binding from individual SCL supporting studies is difficult. To address this problem, this researcher identified all relevant properties from existing studies first, and then thoroughly examined the said properties and proposed the new ones. Finally, the identified and proposed properties were divided into three groups based on the previously defined criteria: 'semantic suitability', 'workflow suitability', and 'expressiveness'.

3.3.1 Semantic suitability

The contracting concepts "Who", "Where", and "What", can be used to define the properties of smart contracts. The parties involved in the contracting process are described by the *Who concept*. In order for an e-contract to be legally binding, it must have at least two parties. A mediator is also frequently included in the contracting process. In a contract, the parties specify their rights and obligations so that if one party exercises their rights, the other party must comply. Contract context is defined by the content of the contract, which influences the roles of the actors, exchange values, and contracting processes. The *Where-concept*, on the other hand, describes the contract's legal and business context. The contract's legal provisions are intended to resolve disputes, while the business context is critical in determining the contracting processes' requirements, according to [5]. Finally, the *What-concept* describes the exchange value, as well as its provision for each contractual party, which includes services, products, and financial rewards. Also specified as exchange values in a contract are service descriptions, such as service type, role, and so on. The process flow in a successful value exchange requires an exchange-value provision, according to [128].

Table 5: Contractual aspects required for legally enforceable SCLs, and associated supporting studies [Publication (I)].

Aspects	Paradigm	Properties	Associated Studies
Suitability	Semantic	The Who-concept	SS23, SS28, SS36
		The Where-concept	SS35, SS42, SS46
		The What-concept	SS20, SS31
	Workflow	Control-flow	SS39, SS41
		Data-flow	SS34, SS45
		Resource-flow	SS29, SS30, SS44
		Exception-handling	SS32, SS43
		Event-Log Imperfection	SS33, SS34, SS37
Expressiveness (*viz. formal-verification*)		Temporal constraints	SS19, SS21, SS25, SS38
		Structural constraints	SS22, SS24, SS26, SS27

3.3.2 Workflow suitability

These patterns describe the fundamental requirements that arise on a regular basis during business-process modeling. Because SCs follow contracting processes, the patterns required for SCL design have been identified in Table 5. Among the patterns mentioned are control flow, data flow, resource flow, exception handling, and event logs. *Control-flow patterns* such as XOR, and AND, are used in the design of imperative models to depict the relationship between activity and flow [144]. The *data-flow patterns*, on the other hand, describe how data are represented and used in business processes to ensure transparency, as well as how data elements interact with one another [60]. Most existing languages currently focus on *control and data flow*, and the *resource perspective* is unfortunately ignored. Resources such as humans, and machines, and the roles they play, must be accurate because they affect the simulation of the in-house process of service consumer and service provider in business collaboration [145]. Exceptions are deviations from standard execution that occur during the course of a business process. 'Integer overflow', 'call stack', 're-entrancy', and other unexpected events are difficult to define. As a result, an exception handler resolves the impact of such events as soon as they are detected. An *event log*, which is a collection of multiple records, represents a sequence of events carried out in a single execution process. The goal of an event log is to uncover useful information about business processes by employing a variety of techniques such as data and process mining [156].

3.3.3 Expressiveness

The expressiveness properties of smart-contract code determine its formal mathematical correctness[128]. Several verification tools, including Oyente, Securify, SmartCheck, and others, are used to detect unintended behavior in SCs [114]. This enables the parties to test the functionality of SCs before publishing them on blockchain platforms. Contract semantics must be verified using formal methods such as static analysis, model checking, and formal semantics, among others. The 'Securify Tool,' for example, is an Ethereum contract security analyzer that symbolically analyzes contract behavior by extracting precise semantic information from code [114]. These tools facilitate learning about the SCs' formal representation, which includes their temporal and structural properties [161]. Contractual events, obligations, and rights, have temporal properties because their execution state varies over time [151]. Each event, such as payment or delivery period, has a specific date and time for action. Contracts, like obligations, have a start date, a due date, and a discharge date. Rights have a beginning and ending date or time, and are activated when specific events or dates occur. The structural properties of SCs govern their functional behavior. To validate the functional correctness of a smart contract, Liu et al., [94] proposed a formal verification method based on Colored Petri Nets (CPN). CPN [11] is a programming language used for system specification, simulation, and design. Statechart is an alternative for defining the structural properties, and constraints, on the sequence of states of a state machine [161].

The critical properties that make a smart-contract legally binding, such as 'suitability' and 'expressiveness', have been identified and listed in Table 5 together with the research that has been done on them.

3.4 Evaluation of SCL Suitability and Expressiveness

As described in Section 3.3, existing SLCs have been evaluated using the contractual-enabling properties identified for SQR 1.2. The SLCs have been scored for each of the ten properties using '+' or '-' operators. The '+' symbol indicates that the SCL has the specified property, whereas the '-' symbol indicates that the property is not present in the corresponding SCL. If it was not possible to identify the properties of a SCL, n/a is used as the

[11] CPN | Home Page

indicator. The final results of the evaluation are shown in Table 6.

- *SRQ1.3*: What obstacles are there in existing SCLs that restrict the attainment of business-contractual objectives?

The purpose of this SRQ is to evaluate the suitability and expressiveness qualities of the SCLs identified in Section 3.2. As has been mentioned earlier, not all SCLs are created equal, and as a result, it has been discovered that many SCLs do not have all of the suitability, and expressiveness, properties that have been identified and proposed as being necessary for the creation of legally-enforceable SCs (see Table 5). As a result, the SCLs selected for SRQ 1.3 have been re- examined, together with the suitability and expressiveness properties from the existing studies. The results of this analysis are shown in Table 6. The reasons for assigning each of the SCL properties a score are also explained. In this manner this researcher discovered that SCLs can be used to draft legally binding SCs in DAO collaborations.

According to the findings in Publication (I), Solidity expressed the *Who-concept* by incorporating the contracting party's address into the *address owner*. Furthermore, control- and data-flow properties are defined due to the imperative paradigm of Solidity. Because data are presented in states encoded with a highly secure cryptography technique and stored at a specific blockchain address, non-domain users have a difficult time understanding the data-flow of SCs. Because Solidity focuses on manipulating low-level blockchain in java-script style, variables and data types are not thought to support the Where- and What-concept. *OnlyBy()* modifiers enable resource ownership, and Solidity handles exceptions in the form of *try/catch* statements only for external function and contract creation calls. Solidity implements events using logs, and once created, a contract can access the log data. Recent attacks, on the other hand, revealed a lack of formal verification in Solidity, despite the fact that the temporal constraint is satisfied by variables such as *uint start-data*, and *uint end-data*, as well as modifiers such as *checkTime()*, and *onlyOnce()*. Aside from resource-flow, other domain-specific languages, including Idris, and Vyper, support Solidity-like properties. Vyper creates a global variable beneficiary by calling the public function on the *public (address)* datatype, and the modifier *raise(reason: str)* returns the reason for the exception. Idris, on the other hand, expresses the *Who-concept* through modifier *mapping (address=>uint)*, with *Raise handlers* to return the exception. Regrettably, Idris does not specify the enforcement of requirements among the processes due to the program's lack of data awareness.

Attacks exploit unsafe patterns such as 'call-to-the-unknown', 'gasless send', 'exception disorder', and others, resulting in financial or other losses. A 'call to an unknown' function is a primitive that invokes and transfers the digital asset. There are exceptions when an execution runs out of gas. This type of exception cannot be handled in imperative languages.

Flint utilizes the address variable to express the *Who concept*, and the *type state*, to satisfy control and data-flow properties. The Who-concept also addresses resource flow by implementing *asset type*, which prevent a class of security flaws in which a smart-state contract represents resources incorrectly. Function modifiers, such as *require*, are used to check the precondition before entering the function body. Flint, on the other hand, does not express properties such as the 'where, and what, concept'.

The Flint language determines resource consumption using the *Wei asset function*. *Wei* can be defined in Solidity as an integer value, rather than a 'dedicated type'. This allows the conversion between number and currency, thus, resulting in an inconsistent state in which the actual balance of a smart contract is incorrect. Formality and 'Pyramid scheme' SCLs, with the exception of structural constraints, satisfy all of Flint's properties. Formality includes inductive data-types for expressing structural properties. The Pyramid scheme, on the other hand, expresses structural properties by combining formal 'intermediate representation' (IR) semantics and dependently typed language.

A Findel contract is made up of a tuple *(D, I, O)*; consisting of a description, an issuer, and an owner. Findel expresses the *Who-concept* by specifying the address of the owner and issuer. The *Who-concept* is described by specifying the contract context in the description field. Formal semantics of Findel are introduced using a rigorously defined 'put-call parity algebra theorem', and 'time-bound primitives', to express 'temporal conditions'. Yet, 'exception handling', and 'event-log primitives', are missing properties to achieve the objectives of SCLs. The *Who-concept* properties are expressed by Michelson's *address* data-type. *Mutez* types allow for resource manipulation that reveal *resource-flow* properties to be restricted. The use of the timestamp Michelson code also satisfies the *temporal-constraint* requirements. To retrieve the current timestamp for an event, the Now(), and Add(), functions are used. Other Tezos blockchain-based languages, such as Liquidity and Fi, meet Michelson's underlying properties while also qualifying the structural properties.

Rholang's syntax, and semantics, are similar to rho calculus, a reflective higher-order variant of pi-calculus. Resource flow is expressed in Rholang by implementing the *phlogiston metric*, which is identical to Ethereum gas. Rholang considers how information is stored and retrieved by way of channels. Marlowe, unlike other SCLs, implements the *Where-concept* as a Haskell datatype. Several of the contracts also include timeouts that specify how they will act. Marlowe implements the *step function* that operates on each constructor of contract type and thus, satisfies the temporal constraints. Pact SCL, stores party addresses in the *keyset variable*, and prefers a declarative approach to complex control-flow, thus making bugs more difficult to write and detect. Sophia articulates concepts such as *Who-concepts*, *Control-Flow*, *Data-Flow*, *Eventlog*, and *Exception Handling*. Sophia employs *address* literals to specify the location of a contract or a party. A contract declares a datatype event to use events that are logged using the *chain.event function*. Contracts can fail with an (uncatchable) exception using the built-in function: abort (reason : string). Fift uses the abort inbuilt function to throw an exception with an error message. Rather than supporting smart contract semantics, Ivy and Typecoin were created with the goal of writing scripting code for cryptocurrency.

Lolisa supports solidity features such as mapping, modifiers, contracts, and address types. Lolisa was designed to formalize Solidity programming languages. The primary functions of Bamboo, Move, and Obsidian are to facilitate resource flow. Bamboo implements the reentrancy-guard when calling recursive functions, and expresses the *Who-concept* by using variable *(address=> uint256)* to identify addresses of the parties. Move's type-system prevents resources from being duplicated, lost, or copied. When a resource is not moved, for example, by deleting the move line, a bytecode verification error occurs (coin). Obsidian uses a type system to ensure that assets are handled correctly, as well as linear types for safe object manipulation. To express *control-* and *data-flow* properties, Obsidian also implements the polymorphic linked list, which is a container for storing transactions. In the current practice of representing Bitcoin contracts as cryptographic protocols, the *Who-concept* and *control-flow* are expressed using opcodes, which are a list of scripts.

BitML expresses *temporal constraints* by choosing a secret and revealing it using the time constraint 't', as well as introducing the symbolic and computation model to describe structural properties. Simplicity seeks to express resource flow using a formal semantics that allows for “fast” (linear time) static analysis of resource consumption. Scilla defines the *address* variable to express the *Who-concept* properties, and the *resource-flow* is expressed using variable *mapping (address => uint)* assets. Integrating the Scilla into the proof assistant of *Coq* enables it to reason about the properties of the security and temporal-constraints. IELE expresses the *control-flow* in which the body of the function contains code organized into labeled blocks. When a branch instruction is encountered, the execution falls through from the last instruction of a block to the first of the next one, or jumps to the beginning of a specific block.

Table 6: Evaluation of SCLs pertaining to business-contractual aspects [Publication (I)].

SCL Types	SCLs	Suitability*								Express-iveness*	
		Semantic			Workflow					(*viz. Formal Verification*)	
		WO	WR	WT	CF	DF	RF	ExH	EII	TC	SC
Domain Specific	Solidity (GL13)	+	-	-	+	-	-	+	+	-	+
	Vyper (GL12)	+	-	-	+	+	-	+	+	+	-
	Idris (SS14)	+	-	-	+	+	-	-	-	-	+
	Flint (SS13)	+	-	-	+	+	+	+	-	-	-
	Formality (GL14)	+	-	-	+	+	+	-	-	+	-
	Pyramid (GL15)	+	-	-	+	+	-	-	-	-	-
	Findel (SS5)	+	+	+	-	+	+	-	-	+	+
	Michelson (GL9)	+	-	-	+	+	+	-	-	-	+
	Plutus-Core (GL10)	+	-	-	-	-	+	-	-	-	+
	Rholang (GL11)	+	-	-	+	+	+	+	+	+	+
	Marlowe (SS16)	+	+	-	-	+	-	-	-	-	+
	Pact (GL2)	+	-	-	-	+	-	-	-	+	-
	Sophia (GL24)	+	-	-	+	+	-	+	+	-	-
	Fift (GL25)	+	-	-	+	+	-	+	+	-	-
	Ivy (GL5)	+	-	-	-	+	+	-	-	-	-
	Typecoin (SS17)	+	-	-	-	+	+	-	-	-	-
Formal Verification	Lolisa (GL26)	+	-	-	+	+	-	+	-	+	+
	Bamboo (GL6)	+	-	-	-	+	-	-	-	+	+
	Mutan (GL19)	+	-	-	+	+	-	-	-	+	+
	Script (SS18)	+	-	-	-	-	-	-	-	-	+
	Move (GL21)	+	+	+	-	+	+	-	-	+	+

	Obsidian (GL27)	+	-	-	+	+	+	-	-	-	-
	IELE (GL23)	+	-	-	+	+	+	-	-	-	+
	Fi (GL7)	+	-	-	+	+	-	-	+	-	+
	Scilla (SS9)	+	-	-	-	+	-	-	-	-	+
	BitML (SS3)	+	-	-	-	-	-	+	-	-	+
	Simplicity (SS1)	+	-	-	-	+	+	-	-	+	-
	Liquidity (GL8)	+	-	-	+	+	-	-	-	+	+
	F-Sol (GL17)	+	-	-	-	+	-	-	-	-	+
	FSolidM (SS15)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	BALZaC (GL3)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
User Friendliness	LLL (GL16)	+	-	-	+	+	-	-	-	+	+
	Babbage (GL18)	+	-	-	-	+	-	-	-	-	+
	RIDE (GL22)	+	-	-	-	-	+	+	-	-	-
	DSL4SC (SS2)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	SmaCoNat (SS10)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Legally Enforceable	ADICO (SS7)	+	+	+	-	+	-	+	-	-	+
	ErgoScript (GL20)	+	+	+	-	-	+	-	-	-	+
	SPESC (SS49)	+	+	+	-	-	+	-	-	-	-
	Reaction-Rule ML (SS1)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	Commit-Rule ML (SS8)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Business Process	DAML (GL4)	+	+	+	-	+	+	-	+	-	-
	eSML (SS4)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	BCRL (SS6)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	QSCL (GL1)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

* *WO* [Who-concept] | *WR* [Where-concept] | *WT* [What-concept]
* *CF* [Control-flow] | *DF* [Data-flow] | *RF* [Resource-flow] | *ExH* [Exception-handling] | *EII* [Event log imperfection]
* *TC* [Temporal-constraint] |*SC* [Structural-constraint]

It is important to point out here that several foundations, such as ADICO, ErgoScript, CommitRuleML, ReactionRuleML, SPECS, BCRL, and eSML are proposed in academic papers that have unfortunately not spawned any further implementation or research efforts. As a result, the Supporting studies for the above mentioned SCLs have not yet been published in the literature. Consequently the properties of these SCLs have been represented with the ‘n/a’ (not-applicable) symbol in Table 6.

3.5 Novel Framework for Designing Legally-Binding SCL

It is clear, based on a systematic and detailed analysis of the existing literature, that none of the SCLs are capable of supporting the creation of legally binding SCs in their current form, without external support such as smart-contract templates or frameworks. Despite the fact that there are alternative solutions for designing semantically correct [33], and formally verifiable [94] contracts, this researcher believes that future SCLs should be designed systematically from the bottom up to ensure the robustness of SCs. As part of this study, this researcher provides suggestions on how to accomplish this.

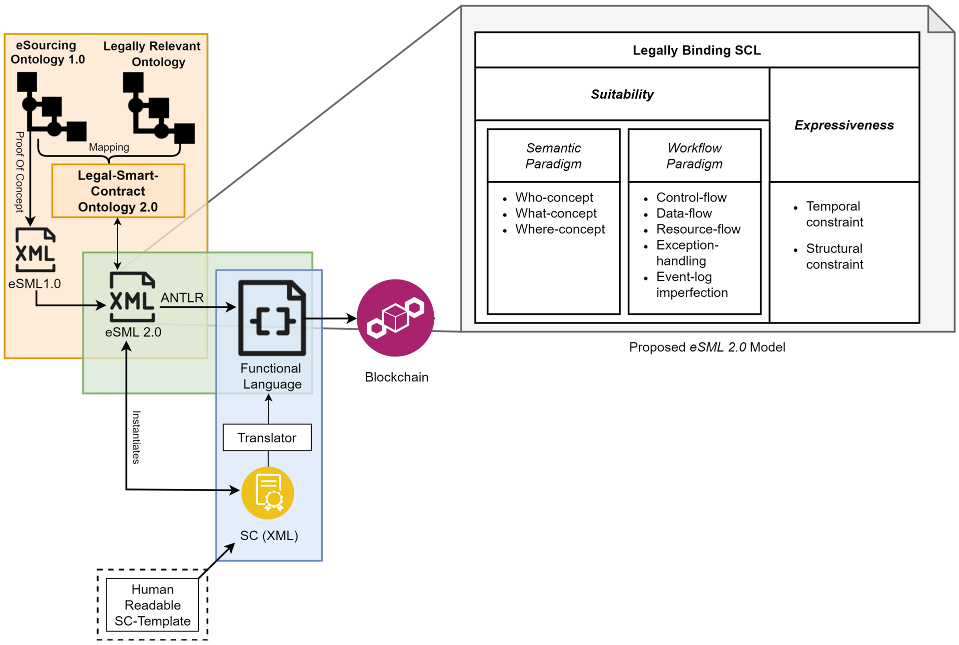


Figure 6: Proposed framework for the development of legally binding smart-contract language [Publication (I)].

The challenges mentioned in this chapter have to be overcome before the full potential of SCs can be realized in DAOs, and other types of business collaborations. According to the current literature, the gaps in the state-of-the-art SCLs can be addressed in two ways: First, by developing graphical SCLs for writing semantically correct smart-contract codes (as shown in Reference [171]). Code that is known to be correct, error-free, and based on predefined procedures, and best practices, can be divided into modules and reused in new solutions. The second approach is to design new SCLs from the ground up while drawing on insights from published literature.

This researcher proposes a novel framework methodology for the development of a legally-binding smart-contract language.

The proposed framework, illustrated in Figure 6, builds on Norta et al.,'s eSML 1.0 [128] by mapping eSourcing ontology 1.0 with legally relevant vocabulary, to create a novel ontology that is semantically rich enough to describe all business collaboration processes (Legal smart-ontology 2.0).

The legally relevant vocabulary is developed using the Protégé tool [12] and the HermiT

[12]Protege | Home Page

reasoner[13]. The newly-developed legal smart-contract ontology 2.0 is then mapped to Norta et al.,'s eSML 1.0 to create eSML 2.0, which will be rich enough in in legal semantic vocabulary, and robust enough to handle any potential temporal and structural vulnerabilities, and thus be deployable across real-world blockchains.

3.6 Chapter Conclusion

This chapter aims to develop a model-driven framework for creating a smart contract language that supports legally relevant and business-collaboration properties. To do so, 45 cutting-edge smart-contract languages designed for business collaboration are investigated. Based on existing research, ten critical properties that make SCs legally enforceable are identified and categorized into three groups: "semantic suitability," "workflow suitability," and "expressiveness." Through systematic analysis it is discovered that none of the current cutting-edge SCLs satisfy all of the suitability and expressiveness requirements. Based on these findings a methodology, and framework, for developing legally enforceable SCLs is proposed. In addition, it is also proposed that the creation of a ontology for contractual business semantics could result in legally binding SCLs. Mapping the proposed semantic ontology onto the contractual choreography language in eSML 1.0 could result in the development of a new functional language for smart-contracting DAO collaborations. This researcher believes that legally enforceable SCLs, when implemented, could make businesses more flexible, participatory, and competitive.

[13]HermiT | Data and Knowledge Group

4 FORMAL SPECIFICATION LANGUAGE

This chapter discusses the work done, and the findings related to, research question RQ2: “How can a formal-specification language be developed for the purpose of legally-binding DAO collaboration?”, and suggests a methodology that can be followed to develop three key artefacts: 1) Legal-smart-contract ontology, 2) a workflow model, and 3) Smart Legal Contract Markup Language.

This chapter investigates the semantic suitability required to define the legal aspects of business contracts [Publication (II)], and workflow suitability required to define the workflow process and patterns [Publication (III)]. This chapter also proposes the vocabularies of semantic and workflow suitability in an XML schema, which enables the specification of legally binding SCs (expressiveness) [Publication (II)].

4.1 Introduction

As discussed in running case 2.3.1: Decentralized system of the automobile supply chain, specifying legally-binding and collaborative smart-contracts is challenging. However, as proposed in the novel framework for designing legally binding SCLs in Chapter 3, this can be accomplished by developing a conceptual model, with the necessary contractual concepts and properties (suitability) to render smart contracts legally binding. In this research, the methodology for developing ontology, workflow model, and SLCML artefacts, as proposed in the framework is followed. It should be noted, however, that the workflow model artefact, is a minor deviation that is used to design workflow properties. An ontology-driven conceptual model is chosen to formalize the contractual, and business-collaboration concepts and properties, because it is an appropriate means to conceptualize the knowledge of a specific domain [99]. Furthermore, research [42] shows that ontology-driven conceptual modeling can be used to overcome inconsistencies and shortcomings in blockchains.

The first contribution of this chapter is the development of the SCL ontology, as presented in Publication (II). The SCL ontology was developed with the protégé tool [95] and verified with HermiT reasoner. Protégé [95] is an open-source ontology editor with a graphical interface for visualizing concept relationships. The classes and properties of the proposed SCL ontology were identified and developed through exhaustive interviews with domain experts, as described in Publication (II).

As the second contribution of this chapter, the SCL ontology was formalized in the workflow model to test the formal verifiability of the ontology in contractual workflow, using the Colored Petri-Net (CPN) simulation tool, this artefact (workflow model) has been presented in Publication (III). The CPN tool can be used to design, develop, and analyze the processing state of SCs in order to track the fulfillment of contractual concepts and properties [93]. Like SCs, the CPN tool consists of states, transitions, code, and tokens and is a graphical representation of a SC. The final contribution of this chapter is the translation of the concepts, and properties, of the ontology, and workflow model, into an XML-based language, referred to as Smart Legal Contract Markup Language (SLCML)[Publication (II)]. The proposed SLCML is a semantically rich, process-driven, and formally verifiable language which allows programmers to configure SCs based on their knowledge and understanding. The SCL ontology has been evaluated using the automotive supply chain running case, (Section 2.3.1) in which it demonstrated the ability to resolve rights-and-obligations conflicts between collaborating parties. It should be noted that the SCL ontology is also evaluated during workflow execution of SCs where the formal verifiability of the ontology is tested. The SLCML is evaluated in Chapters 5 and 6.

4.2 Multi-Tiered Contract Ontology

This researcher has extended the set of concepts and properties of the SLC ontology, by taking into account previous work on the collaboration model. This research extends the set of concepts and properties for the SCL ontology, taking previous work on the collaboration-model into account [128]. This researcher's previous work on specifying and verifying harmonized B2B process collaborations [125] defined the eSourcing framework. Based on the concept of eSourcing, the eSourcing ontology [128] was designed to configure collaborating parties and their services in a decentralized, contractual collaboration model. Unfortunately, the eSourcing ontology lacked legally relevant contractual properties when compared to the SCL ontology. In the SCL ontology and the SLCML, a contract includes the legal elements of contractual collaboration such as rights, obligations, and performances. Individual rights are fundamental normative regulations that govern what is permissible, or owed, under a legal system, social convention, or ethical theory [87]. Contract obligations are the legal responsibilities of each party to a contract agreement. The fulfillment of the parties' contractual obligations are referred to as "performance of the contract." In this section, the legal aspects of the proposed SCL ontology are briefly discussed. The collaborative aspects of the proposed ontology are described in the section that follows.

Because contracts can be of various types, the realm and range of each type varies greatly. As a result, it is difficult to express the entire spectrum of contracts in a single ontology because the latter is too large and diverse to be useful. To capture the full range of business-related contracts within a unified model, a multi-tiered contract ontology is proposed. This multi-level ontology progresses from abstract, to specific meta data definition, and then to stratification.

Two layers of the multi-tier SCL ontology are depicted below; other extensions and layers are possible. The upper core layer depicts the broad configuration of SCs that are applicable to the majority of the common types of contracts. As shown in Figures 7 and 8, fundamental concepts such as rights, obligations, and roles, are considered to be building blocks for defining all types of business contracts. The specific domain layer is made up of various contract-type ontologies, such as employment contracts, sale of goods, sale of services, and so on. Each contract-type inherits all of the fundamental characteristics of the upper-layer, and then specializes in the knowledge specific to the contract domain, as shown in Figure 9.

4.2.1 Upper core layer of smart contracts

The upper core layer of legally-relevant smart-contract DAOs is depicted in Figures 7 and 8, and is explained using a business-case scenario. Using the running case scenario of CarMan described in Section 2.3.1, the classes belonging to the upper-level smart-contract ontology (illustrated in Figures 7 and 8) can be explained as follows. Assume a running-case scenario from Section 2.3.1, in which SupTr and SupSt promise to supply CarMan with tyres and steering wheels, and CarMan promises to pay money in return. A promise is a statement of intent to do something, or to do certain things, such as provide tyres and steering wheels in exchange for payment. When promises are made with the intention of proving them in court, they become *legal obligations*. The legal testimonials of the promises (*obligations)* come from the contracting parties (*actors*), and are specified in the contracts. These testimonials include details about how the *obligations* are composed, accepted limits, and *performance* measures. The *actors* who carry out the roles specified in SCs are the offeror (the party that makes an offer to purchase something), offeree (the party that receives an offer), and mediators. In the automotive supply chain

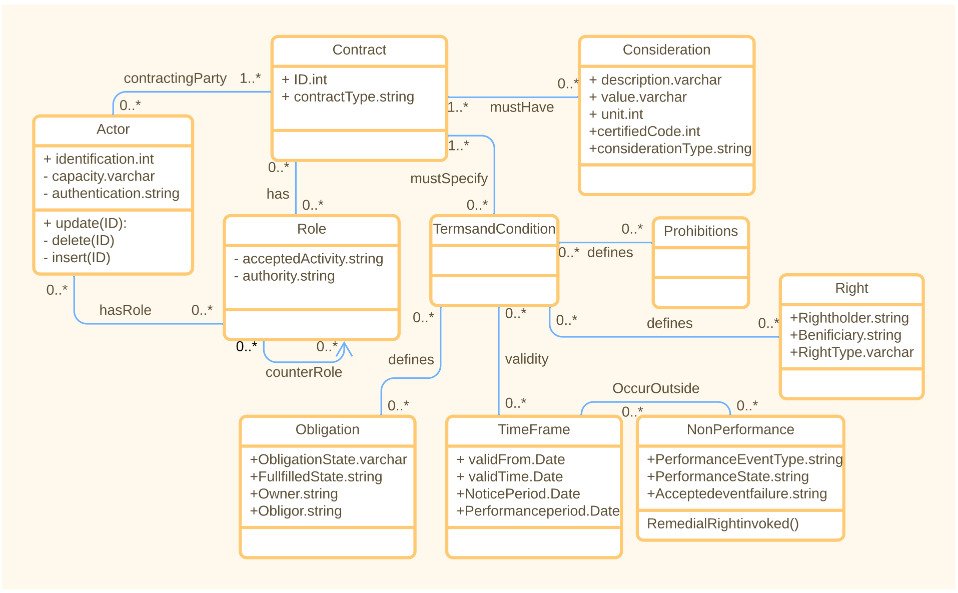


Figure 7: Outline for the upper-level smart-contract ontology.

running case, CarMan is a buyer who offers a purchase contract to a supplier, and in legal terms is known as an offeror. SupTr, and SupSt are service providers (suppliers) who are the recipients of an offer, and are therefore known as offerees. CarMan makes an offer to SupTr and SupSt to buy the tyres and steering wheels, and in return, agrees to pay money. According to [70], a smart contract is legally enforceable if the contracting parties have the necessary *capacity or competence* to enter into it. If a party is unable to understand the contract, or is presumed to be unable to understand the contract, that party lacks the competence or capacity to enter into a contract. A person who lacks legal capacity, such as someone who is insane, or under a certain age, may be deemed legally incompetent to enter into a contract. Companies such as CarMan, SupTr, and SupSt, who collaborate in DAOs must be legal entities. The legal status of DAOs in Wyoming was recently established [14].

A *consideration* is a negotiated benefit that motivates a party to enter into a contract. In exchange for *performance*, a valuable consideration (at least in the eyes of the parties) must be exchanged. For example, tyres and steering wheels, are *considerations* for which CarMan, SupTr, and SupSt have signed contracts. The delivery of tyres and steering wheels, as well as the transfer of ownership, through the payment of money, constitute the *performance* of the sales contract. Considerations can also be as simple as a promise to repair a leaking roof or a promise not to do something [15]. A similar consideration occurs if CarMan signs a contract with SupTr under which CarMan agrees not to order tyres other than Goodyear and SupTr pays CarMan $500 per year for adhering to this agreement. The sellers' promise; the sale of tyres and steering wheels, is a contractual *obligation* that is fulfilled when the actual business activities of supplying tires and steering wheels for money are carried out. CarMan is a beneficiary, or claimant who receives the consideration, or is the individual to whom the business operations are performed. Finally, SCs specify the

[14]DAO | Legal status
[15]Consideration | Legal Definition

terms and conditions for the delivery of agreed-upon services. Contractual performance is typically carried out in accordance with the contract's terms and conditions. If the performance is not completed within the expected *timeframe*, or is completed in an insufficient manner, the obligation state becomes unfulfilled. The occurrence of a *non-performance* event gives the promised party certain pre-agreed-upon rights. Assume the SupTr fails to deliver the tyres to CarMan according to the agreed-upon terms. CarMan may seek restitution in the form of a penalty or interest, or he may prefer to terminate the contract as specified in the contract. However, CarMan, could avoid retaliation and punishment, and instead resolve the conflict peacefully with mutual agreement on how to proceed. The service provider must fulfill any type of remedy (reconciliatory promise) requested by the CarMan. Once the reconciliatory promise is made, the initial commitment is considered complete.

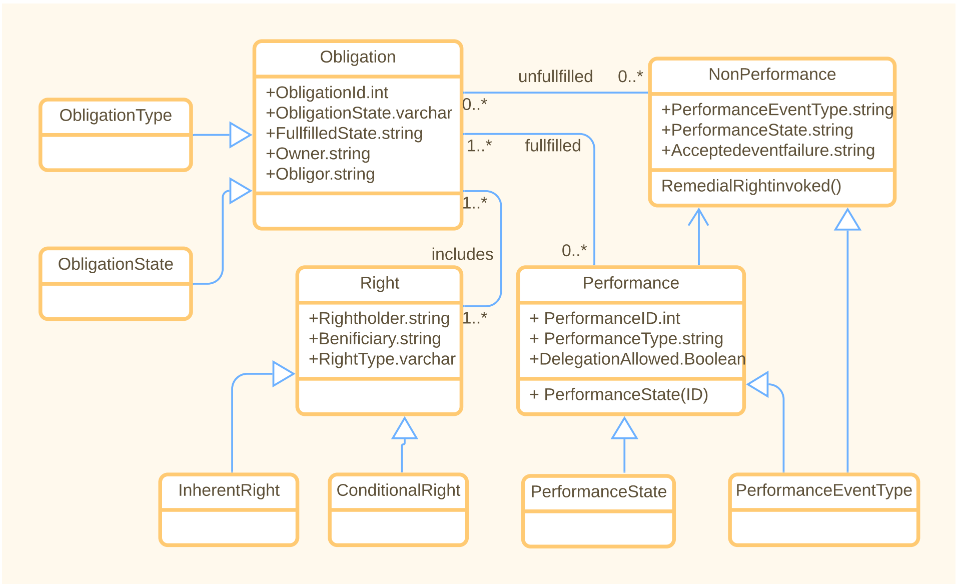


Figure 8: Rights and obligations [Publication (II)].

The preceding scenario is a simple case study that demonstrates how obligations can result in additional obligations and rights. Similarly, rights can create new obligations. The following section deals with obligation types extracted from the upper-layer ontology.

4.2.2 Specific domain layer

Contract statements can be, according to [87, 178], informative, declarative, or performative. Informative statements recognize a variety of details, such as the identities of the parties, the applicable law, the subject matter of the contract, and so on. A declarative statement expresses an intention or a condition that cause the state to change when the specified conditions are met. Declarative statements are usually classified into three types: *rights*, *obligations*, and *prohibitions*. *Obligations* are contractual statements in which the obligation owner, the party who receives the obligation, and the obligor, or debtor, the party who performs the obligation, are both mentioned. The obligor, or debtor, is only required to execute the obligation condition once in each execution of the contract. *Rights*, like obligations, have holders and beneficiaries, with the holders carrying out the duties. The exercise of a right is optional, and it may be carried out under certain conditions based on the fulfillment of obligations. *Prohibitions* are statements that specify which actions should not be taken or are unacceptable to either one or both parties.

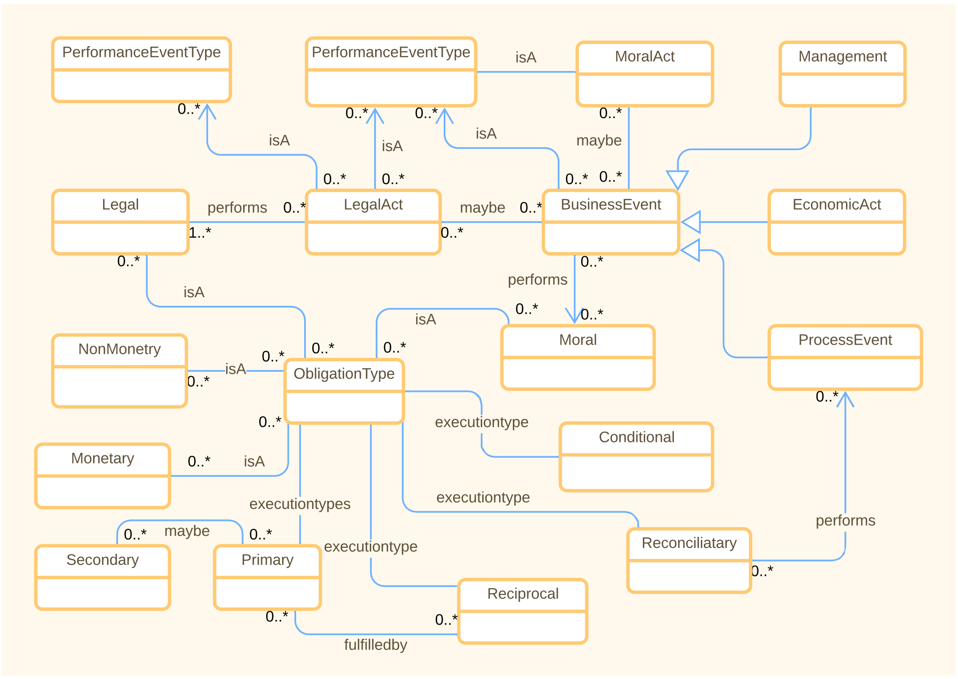


Figure 9: Specific domain layer.

Oligations must be tied to both *performative* and *non-performative events* in order to fulfill the former. Obligations are classified as *primary*, *reciprocal*, *conditional*, or *secondary*, based on the nature of their execution, as illustrated in Figure 9. If the primary goals of the contract are met, the *primary obligations* of the contract are also met. For example, When SupTr delivers the tyres in accordance with the contract, or when CarMan accepts, and pays for the tyres as ordered, SupTr's and CarMan's primary obligations are met. Although the *reciprocal obligation* is the primary obligation, the counterparty must also perform the reciprocal obligation in response to the execution of the primary. The CarMan's obligation to pay, and the SupTr's obligation to deliver, are mutually exclusive. One of CarMan's primary responsibilities is to pay SupTr. A conditional obligation, however, does not have to be triggered in the normal course of events. This category includes the vast majority of redress rights and obligations. For example, if CarMan does not receive the tyres and steering wheels within a certain period, CarMan may seek late delivery compensation. As a result, the service provider is required to deliver the goods, in addition to paying an additional penalty fee. Finally, a *secondary obligation* is a part of a primary obligation that can be activated for additional commitment. SupSt and SupTr, for example, are committed to offering packaging services (for example; shipping options for packages), despite the fact that they are not legally required to do so.

Obligations can be segregated into *legal*, *business*, and *ethical* obligations based on the context of the obligation that requires a specific type of performance. Every statement in a business contract is legally binding, and carries legal consequences. Nonetheless, the legal obligation category is proposed in order to differentiate those obligations that require some specific legal actions to be carried out to fulfill them. The legal acts may or may not be part of the business management process. Likewise, business obligations are legally binding. However, this researcher proposes that this term categorizes all those obligations that are specifically related to business performance. ‘Business obligations can be classified into two types: *monetary* and *non-monetary*. *Monetary obligations* are those with economic or financial consequences, such as late-payment penalties. *Non-monetary obligations* are those commitments that have reputational consequences. While not every business obligation is monetary in nature, business obligations do have legal consequences.

Execution of business processes are required for commitments such as CarMan sending orders to buy steering wheels following acceptance of the contract, or SupSt arranging for the carrier (shipping and transport), and notifying CarMan, and so on. Tyre replacement, logistics carrier arrangements, and other obligations between CarMan and SupSt have no economic implications, and may be considered to be non-monetary obligations. Moral or ethical obligations may not be severely binding but are more practically or socially expected obligations [16]. For example, though it maybe CarMan’s obligation to pick up the goods from the seller’s premises, he may request the seller (SupTr or SupSt) to help in arranging the transportation or may require some other help, such as, arranging for payment of customs duty. The seller (SupTr or SupSt), though he may not be legally bound to assist the buyer, is morally bound to aid the buyer should the buyer request such help.

4.3 Rights and Obligations Monitoring

As described in Section 3, the absence of formally verifiable SCs has the potential to result in significant financial loss. This can happen because SCs are unaware of their own processing state. And, in cases of contractual disputes, tracing how an SC as been executed is difficult, time consuming, and expensive.

Several formal verification methods have been proposed to ensure contract security [168, 104, 100]. Formal methods use mathematics to find unknown vulnerabilities in a contract. In this chapter, the CPN tool is used to validate ontological properties in a contractual workflow process.

The SCL ontology is formalized in Colored Petri Nets (CPNs), which can be used to design, develop, and analyze the processing state of SCs [93]. The CPN simulation tool can also track the fulfillment of contractual properties. The workflow model represents the definition of business processes and workflow patterns based on an ontology. Coloured Petri Nets, like smart contracts, consist of states or transitions, code, and tokens and is a purely formal system representation of a SC.

This thesis extends the existing formalized smart-contracting lifecycle [121], [129], and [123], which comprise the business collaboration model but does not have legally binding properties. Smart contracting language ontology concepts and properties are mapped to existing smart-contracting lifecycles in order to monitor the related contractual fulfillment process throughout the entire collaboration model. The updated CPN lifecycle model is known as the *SLC lifecycle model*. The SLC lifecycle begins with the configuration of a business network model (BNM), which is essentially a blueprint for inter-organizational collaboration that includes a legally binding template contract that matches service types to organizational roles. The template allows for the identity and reputation of the contracting parties, and the services they are contracting for, to be determined quickly, and in a semi-automated manner.

[16]International chamber of commerce | Home

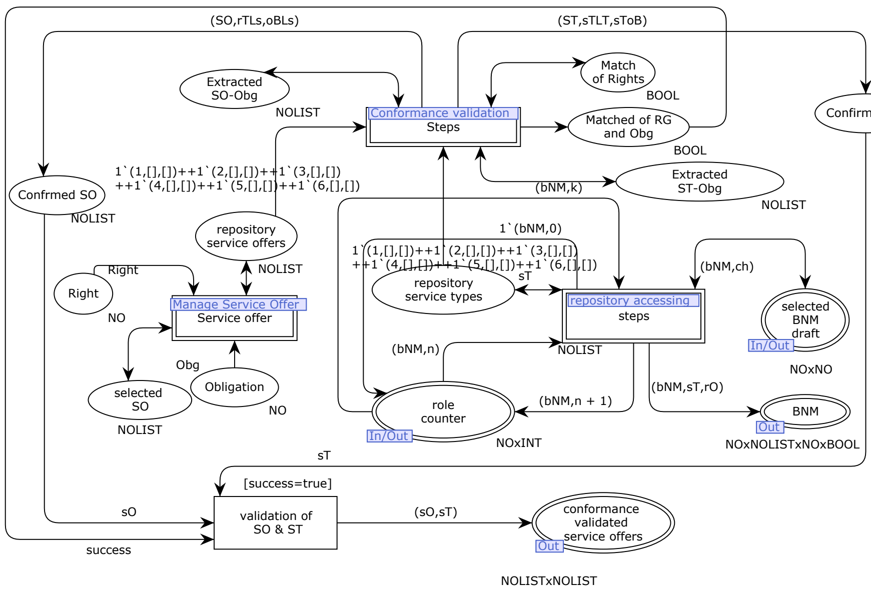


Figure 10: Rights and obligations selection in BNM.

The *SLC lifecycle* shown in Figure 10 illustrates a nested module, labelled *Repository Accessing*, in which a user enters the service type with rights and obligations over time in the *BNM selection*. The same assumptions apply to the service offer repository in the *Manage Service Offer* module. Finally, the *Conformance Validation* module of SLC lifecycle model is developed to meet the BNM draft specification's validation of service offers against the selected service type. The details of each module are provided in the following sections.

4.3.1 Repository accessing

In Figure 11, it is assumed that a contracting party inserts the rights, roles, and obligations for service types. The contracting party first chooses a BNM from a pool of previously stored BNM drafts. The contracting party then adds the rights and obligations to the *Manage Service Type* module, which is then stored in the *repository service types* state. Additionally, selecting a ´BNM draft' for validating service offers and roles to be filled with rights and obligations is part of the actual BNM-selection process. The rights and obligations that must be filled out in the *Manage Service Type* module are listed below.

4.3.2 Manage service type

The *Manage Service Type* module, shown in Figure 12, is the starting point for the actual repository of service types. The list of 'service type' rights and obligations in the repository is initially empty. The contracting party begins by choosing the ´service type Id' from the *Choose ST* transition. This triggers the *Insert right* and *Insert obligation* transitions, which insert the rights and obligations into *Selected ST* at the same time.

To delete the inserted rights and obligations, user can select the *Delete right* and *Delete obligation* transitions. The same assumption is made when selecting the rights and obligations for a service offer in the *Manage Service Offer* module.

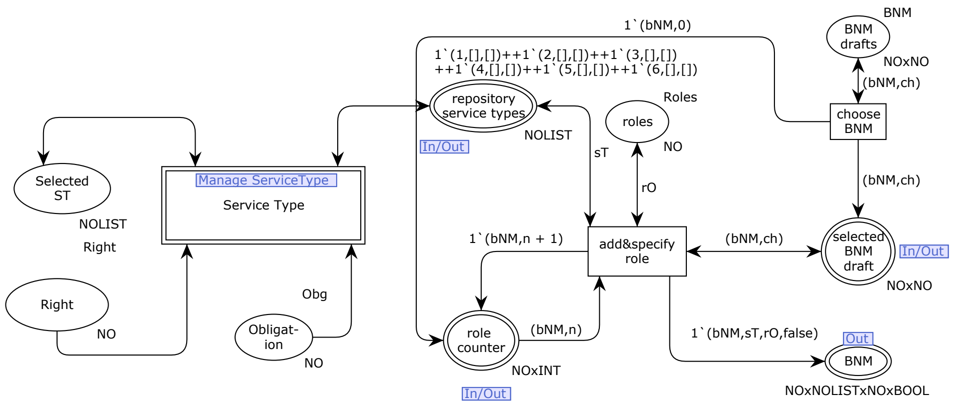


Figure 11: Repository of service type with rights, roles, and obligations.

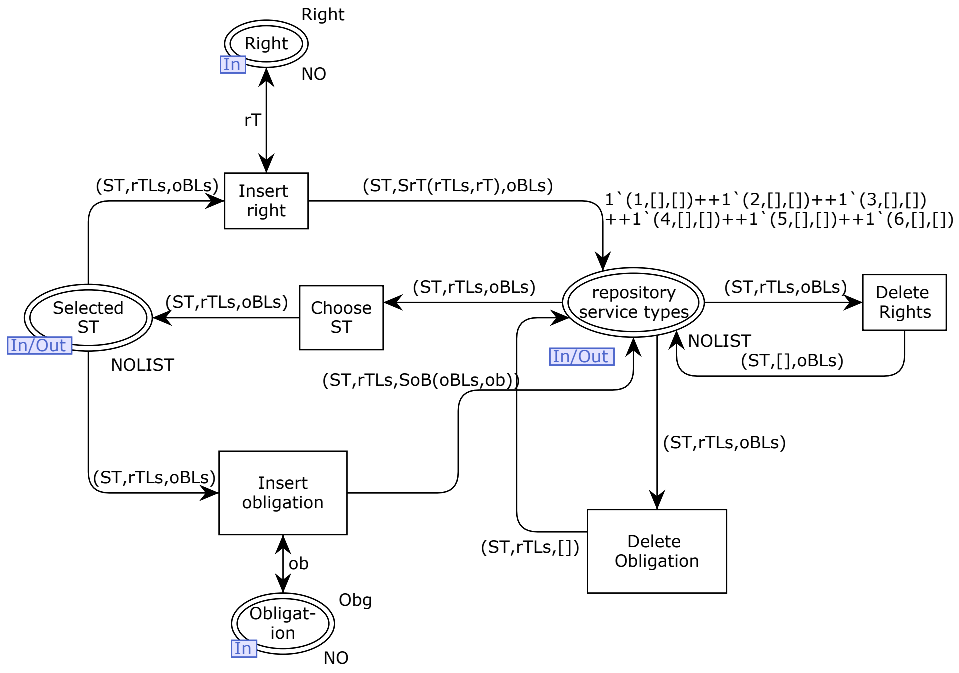


Figure 12: Insertion, deletion of rights and obligations in service types.

4.3.3 Conformance validation

As illustrated in Figure 13, a *´conformance validation* is required before being considered as a service offer for finalizing the prototype SC. Validation is performed on the selected ‘service offer’ and ‘service type’ from the BNM-repository. The properties of the service type are inherited by the selected right and obligation properties. As a result, rights and obligations are extracted from the states *Repository Service Type* and *Repository Service Offer*. A service offer that is matched with a service type is also saved in the *confirmed SO* and *confirmed ST*.

Further, the concepts and properties of ontology and workflow model is translated into XML language.

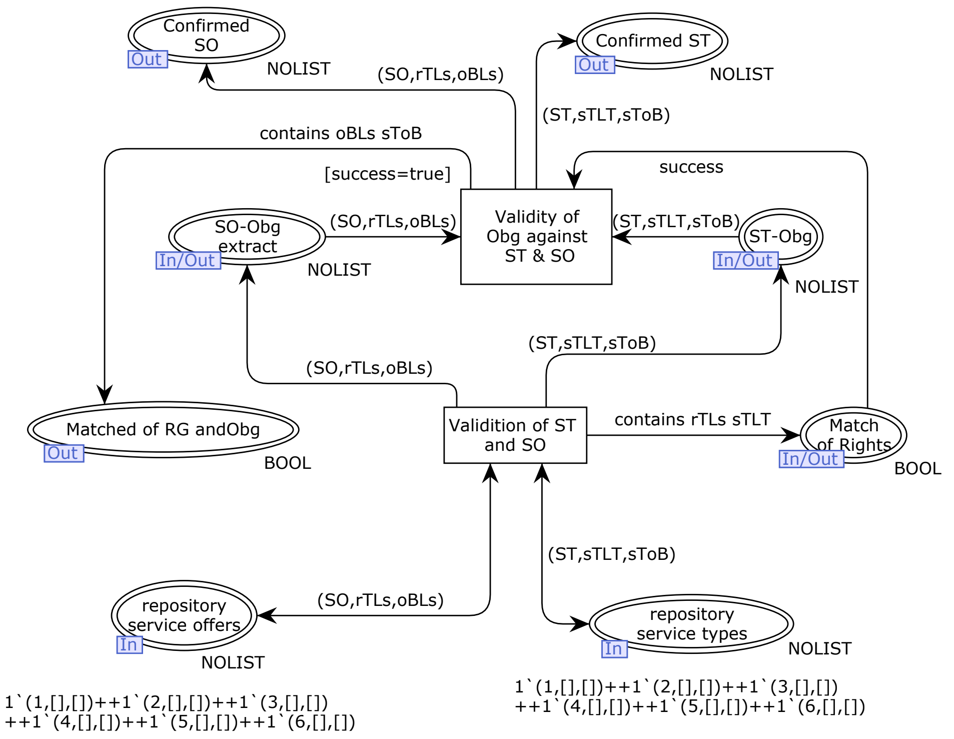


Figure 13: Conformance validation of service offers and service types.

4.4 SLCML: A Contract Specification Language

This section describes elements of the Smart Legal Contract Markup Language, based on legally-smart contract ontology 2.0.

The extended proposed SCL ontology (legally-smart contract ontology 2.0) incorporates the legal concepts and properties for contractual collaboration in business DAOs. The SCL ontology has been formalized into a workflow model using the CPN tool. This workflow model (SLC lifecycle model) contains both semantic and workflow properties and is used to check for the fulfillment of ontology properties throughout the workflow processes. The workflow model is translated into machine-readable language, referred to in this thesis, as Smart Legal Contract Markup Language (SLCML).

For a smart-contract to be legally binding, it must be created using a programming language that contains all of the necessary legal elements. To address this issue, this researcher collaborated with a lawyer [17] to improve the existing eSourcing ontology by adding legally-relevant concepts to eSourcing Ontology 1.0.

Legal-smart-contract Ontology 2.0 was developed by mapping legally relevant ontology against eSourcing ontology 1.0. This ontology was used to develop eSML 1.0, an eSouring Markup Language designed to provide answers to three critical contractual questions: Who? Where? and What? The “Who-concept” describes the parties engaged in the

[17]Alexander Wulf contributed to this thesis by supporting the creation of the smart contract law ontology with his legal expertise. He did not contribute towards the written text of the thesis

contracting process. The “Where-concepts” distinguish the fundamental aspects of the context of an electronic-contract, and “What-concepts” define the exchanged values and their associated conditions. The primary goal of eSML 1.0 was to enable smart-contract collaboration within the eSourcing domain.

This researcher used Liquid Studio [18] to translate the extended concepts and properties of the SCL ontology into the eSML language, which is an XML schema editor for creating XML documents. This translation resulted in the creation of eSML 2.0. The contract specification language (SLCML) is an expanded version of eSML 2.0. As this study is only concerned with the expanded version of SLCML, a link is provided to the complete SLCML schema [19].

The SLCML schema of the upper-level smart contract is presented in Section 4.4.1, and the the schema for defining domain-specific contractual properties in discussed in Section 4.4.2.

4.4.1 Upper-level smart-contract definition

The legal elements described in the upper layer of legally relevant smart-contract DAOs are defined in the code extract shown in Listing 1. As discussed in Section 4.2, the element role in Line 4 defines the role of the parties to the contract; the buyer and seller. Line 5 of Listing 1 defines the contractual considerations as well as the variable types. The value of `minOccurs` and `maxOccurs` in Line 5, represents the amount of consideration required for a legally binding smart contract. Line 6 defines the `terms_and_conditions` element, which specifies the smart contract’s terms and conditions. The contracting party’s description is defined on line 7 of Listing 1, followed by the custom type `company_info`, which includes the contracting party’s name, type of legal organization, and company contact information.

```
1  <xs:element name="contract">
2      <xs:complexType>
3          <xs:sequence>
4              <xs:element name="role" type="variables_def_section" minOccurs="0" maxOccurs="unbounded"/>
5              <xs:element name="consideration" type="variables_def_section" minOccurs="1" maxOccurs="unbounded"/>
6              <xs:element name="terms_and_conditions" type="terms_and_condition_definition" minOccurs="0" maxOccurs="unbounded"/>
7              <xs:element name="party" type="company_info" maxOccurs="unbounded" />
8              <xs:element name="mediator" type="company_info" minOccurs="0" maxOccurs="unbounded" />
9          </xs:sequence>
10             <xs:attribute name="contract_id" type="xs:ID" />
11             <xs:attribute name="global_language" type="xs:string" />
12             <xs:attribute name="web_service_uri" type="xs:string" />
13     </xs:complexType>
14 </xs:element>
```

Listing 1: Upper layer of the smart-contract schema.

[18] Liquid Studio | Home

[19] shorturl.at/uBHR6

Listing 2 displays the rights, prohibitions, obligations, and time-frames defined by the custom-variable: *terms_and conditions_definition_type*. The code extract in Listing 2 is part of the terms and conditions that define the rules and regulations governing the parties' performance, as discussed in Section 4.2. Line 3 defines the rights of the elements, as well as the custom type, (the `right_type`), which allows the parties to customize the type of rights. `minOccurs` and `maxOccurs` indicate that parties must choose at least one right. Prohibitions and definitions of the prohibitions that may apply to the terms and conditions, are described in Line 4. Line 5 of Listing 2 specifies the obligations as well as the `obligation_category`, which allows the parties to configure multiple obligations. Finally, the `time_frame` is defined in Line 6, which indicates when the terms and conditions will expire.

```
1 <xs:complexType name="terms_and_conditions_definition">
2     <xs:sequence>
3         <xs:element name="right" type="right_type" minOccurs="1" maxOccurs="unbounded" />
4         <xs:element name="prohibitions" type="xs:string" minOccurs="0" />
5         <xs:element name="obligation" type="obligation_category" minOccurs="1" maxOccurs="unbounded" />
6         <xs:element name="time_frame" type="variables_def_section" minOccurs="0" />
7     </xs:sequence>
8 </xs:complexType>
```

Listing 2: Schema definition of terms and conditions.

The `variables_def_section`, is a common variable attribute defined in Listing 3 that contains properties for all SLCML variables, both simple and complex. The string data items necessitate the definition of the `string_type`. The role of the contracting party, for example, could be specified as a `string_type`. The `boolean data type` is required for the definition of boolean contract data items. For example, the `boolean data type` determines whether or not the contract is legally binding. The `integer data type` is used to store contract-id and consideration values. Special data types, such as `money_type` and `event_type`, define specific contractual activities. For example, the `money_type` specifies the amount of money in a specific currency, whereas the `event_type` specifies the type of event that may occur during the contract.

```
<xs:complexType name="variables_def_section">
        <xs:sequence maxOccurs="unbounded">
                <xs:choice>
                        <xs:element name="string_var" type="string_type" />
                        <xs:element name="real_var" type="real_type" />
                        <xs:element name="integer_var" type="integer_type" />
                        <xs:element name="boolean_var" type="boolean_type" />
                        <xs:element name="date_var" type="date_type" />
                        <xs:element name="time_var" type="time_type" />
                        <xs:element name="event_var" type="event_type" />
                        <xs:element name="money_var" type="money_type" />
                <xs:element name="external_resource_reference_var" type="external_resource_reference_type" />
                <xs:element name="list_of_events_var" type="list_of_events_type" />
                <xs:element name="list_of_strings_var" type="list_of_strings_type" />
                <xs:any namespace="targetNamespace" />
        </xs:sequence>
    </xs:complexType>
```

Listing 3: Common variable attributes.

4.4.2 Obligation-type definition

The `obligation_category` consists of the `obligation_type`, `obligation_state`, `performance` and `non-performance` specified in Listing 4. The element `obligation_type`, along with custom variable `obligation_type_definition`, is specified in Line 3; by which several obligations are configured. The `obligation_state` is defined in Line 4, to monitor the contract fulfillment process through which an obligation can pass. In the code example of Listing 4 the definition of `obligation_type_definition` is omitted. The obligation state depends on the performance and non-performance conditions defined in Line 5.

```
1 <xs:complexType name="obligation_category">
2    <xs:sequence>
3       <xs:element name="obligation_type" type="obligation_type_definition" minOccurs="1"/>
4       <xs:element name="obligation_state" type="obligation_state_definition" minOccurs="1"/>
5       <xs:element name="performance" type="variables_def_section" minOccurs="1" maxOccurs="unbounded" />
6       <xs:element name="non-performance" type="variables_def_section" minOccurs="0" maxOccurs="unbounded" />
7    </xs:sequence>
8 </xs:complexType>
```

Listing 4: Schema of obligations category.

Listing 5 is an example of an obligation type from which the parties can create at least one, and possibly more, obligations. The legal obligation is defined on line 3, along with the string variable type. Business obligations have both monetary and non-monetary implications for which `monetary` and `non-monetary` elements are defined in Lines 4 and 5. Line 6, on the other hand, specifies both the string type and the moral obligation. The remaining obligations are defined in a similar manner, as shown in Listing 5.

```
1 <xs:complexType name="obligation_type_definition">
2         <xs:sequence>
3             <xs:element name="legal" type="xs:string" minOccurs="0" />
4             <xs:element name="monetary" type="xs:string" minOccurs="0" />
5             <xs:element name="non-monetary" type="xs:string" minOccurs="0" />
```

```
            <xs:element name="moral" type="xs:string" minOccurs
    ="0" />
            <xs:element name="Primary" type="xs:string"
    minOccurs="0" />
            <xs:element name="Secondary" type="xs:string"
    minOccurs="0" />
            <xs:element name="Conditional" type="xs:string"
    minOccurs="0" />
            <xs:element name="reciprocal" type="xs:string"
    minOccurs="0" />
            <xs:element name="reconciliatory" type="
    business_event_types" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
```

Listing 5: Schema of the type of obligation.

4.5 Chapter Conclusion

The ontological concepts and properties required to create legally binding DAOs are explained in this chapter. Previous work is expanded by adding the legal aspect of DAO collaboration to previously developed eSourcing ontology, which is critical for specifying legal DAOs. The HermiT reasoner was used to to check the correctness of the proposed SCL ontology. The SCL ontology is formalized in the workflow model using the CPN tool in order to track fulfillment of contractual properties of ontology throughout workflow processes. The workflow model represents the definition of business processes and workflow patterns, as knowledge based on ontology. For the feasibility study, the concepts and properties of the SCL ontology together with workflow model were translated into SLCML; a machine-readable language. SLCML is a more complex version of eSML, and only the extended parts of SLCML that are not part of the eSML foundation, were covered in the discussion. A business case from the dairy supply chain will be used to evaluate the SLCML, as discussed in the next chapter.

5 APPROACH FOR TRANSLATING SLCML-BASED SCs TO SOLIDITY

5.1 Introduction

This chapter addresses research question RQ3 (Section 1.2) and describes thesis contribution 6 (Section 1.3). Initially this researcher aimed to develop a translator based on the proposed framework for translating contracts written in SLCML (eSML 2.0) into functional language (blockchain executable language). However, further research revealed the existence of a translator known as Caterpillar. It should be noted, however, that Caterpillar translates only business processes modeled in BPMN into Solidity, and does not accept SLCML schema. Therefore, in this chapter, patterns and transformation rules that deviate from the original framework, are proposed for translating SLCML code to a choreography model, and then implementing Solidity smart contract code.

Section 5.2 demonstrates SLCML evaluation by providing SLCML code examples that stem from a dairy supply chain running case introduced in Section 2.3.2. The complex coordination effort between collaborating parties is facilitated by specifying the inter-organizational contract collaboration with SLCML. The SLCML can be used to configure semantically correct legally binding SCs rather than drafting blockchain executable code. As a result, the patterns and transformation rules are provided in Section 5.3 to translate SCs from SLCML to Solidity (thus satisfying SRQ 3.2). Section 5.4 discusses the feasibility of the SLCML-based SC translation approach to Solidity. This research is described in detail in the Publication (IV).

A number of specification languages including SPESC and Symboleo, have been proposed to configure legally binding SCs as described in Section 3.1. However, a translation approach from a specification language to blockchain-executable code has not been researched. As a result, the research presented in this Chapter fills a gap in the existing literature by answering the research question "How can a BPMN choreography model be built to translate an XML-based contract to blockchain-executable language?"

The proposed translation approach is based on research in [20] and [64], from which this researcher borrows concepts and proposes new patterns and transformation rules that support SLCML vocabularies. The concepts from research [20] are used first to convert the XML choreography model to BPMN, and then the concepts from research [64] are used to translate the choreography model to Solidity. After which, the proposed translation approach is evaluated by generating the Solidity code from the contract written in the proposed language.

5.2 SLCML Instantiation

This section discusses SLCML instantiation while configuring the rights and obligations for the dairy supply chain running case using the SLCML schema from the previous chapter. Listing 6 defines the fundamental contractual elements required for any legally binding business-oriented smart contract. To resolve any conflict, the producer (factory) and distributor create a smart contract with a unique ID that cannot be changed during contract enforcement. Lines 2 and 6 contain the public keys for the producer and the milk distributor respectively. The names of the parties to the contract are specified on lines 3 and 7. Lines 4 and 8 define the contracting parties' roles, namely: producer as a service consumer and distributor as a milk supplier. On line 10, the contract consideration (milk) for which the parties have agreed to a contractual relationship is listed. The terms and conditions include the obligations and rights are outlined in Listing 7 and Listing 8.

```
<contract contract_id="Id1">
        <party address="03 m6">
            <name> Producer</name>
            <role> Service consumer </role>
        </party>
        <party address="31 x7">
            <name> Distributor </name>
            <role> Milk supplier </role>
        </party>
        <consideration> Milk </consideration>
            <terms_and_conditions/>
        <obligation/>
        <right/>
        <prohibitions/>
        <terms_and_conditions>
    </contract>
```

Listing 6: Contract instantiation for the dairy supply chain.

Listing 7 depicts a producer's commitment to compensate a distributor for milk. The obligation has a name and a unique ID that is used to track performance, and it is classified as a monetary obligation because it deals with economic or financial consequences. Line 3 begins the obligation state, indicating that the producer collects milk in accordance with the orders and is required to pay the distributor money. The producer is the obligor, and he or she is responsible for carrying out the obligation stated in Line 6. Line 5's obligations benefit the distributor, and it is assumed that no intermediaries or arbitrators are involved, as indicated by line 7. The producer is expected to act by paying money, and the to-do obligation (line 10) has legal consequences. Line 12 implies the obligations for which the producer and distributor sign contracts (Act 1); the producer receives milk from the distributor. The performance type (line 13) refers to the amount of money that must be transferred from the producer's wallet address to the distributor's wallet address. In addition, the performance object (line 14) is defined as a qualified purchase for which a specific amount is compensated within a specific time frame. Line 15 specifies the purchase-payment plan, while the rule conditions specify the payment time limit. Finally, the obligation is amended to include a mention of the existence of a late payment remedy (line 17). If the producer fails to pay the money within the specified time frame, the producer is required to transfer a certain monetary amount (interest for late payment) to the distributor.

```
1  <obligation_rule tag_name ="paying_invoices" rule_id ="0001"
2  changeable ="false" monetary ="true">
3  <state> enabled </state>
4  <parties>
5     <beneficiary> Distributor (31 x7 ) </beneficiary>
6     <obligor> Producer (03 m6 ) </obligor>
7     <third_party> nil </third_party>
8  </parties>
9  <obligation_type>
10    <legal_obligation> to-do </legal_obligation>
11 </obligation_type>
12 <precondition> act1 (signed)& Milk (transferred) </precondition
       >
13 <performance_type> payment (03 m6,31 x7, buy) </
       performance_type>
```

```
<performance_object> invoice ( buy, amount)<performance_object>
<rule_conditions> date ( before delivery of milk) </
    rule_conditions>
<remedy>late_payment_interest (amount,03 m6 ,31 x7)</remedy>
</obligation_rule>
```

Listing 7: Paying milk obligation illustration.

The obligation intersects with provisions in the Listing 8 code extract. Because the parties' rights and obligations are intertwined, if one party asserts its rights, the other must comply. The rights have a beneficiary who can benefit from them, and an obligor who can enable them, as in Listing 7. If the producer receives poor-quality milk, they have the right to demand that it be replaced. As a result, the distributor will have to replace the milk.

```
<right_rule tag_name ="milk_replacement" rule_id ="0002"
changeable ="true" monetary ="false">
<state> enabled </state>
<parties>
   <beneficiary> Producer (03 m6 ) </beneficiary>
   <obligor> Distributor (31 x7) </obligor>
   <third_party> nil </third_party>
</parties >
<right_type>
   <conditional_right> claim </conditional_right>
</right_type>
<precondition> act1 (signed)& Milk (transferred)</precondition>
<performance_type> replace (poor-quality milk) </
    performance_type>
<action_object> milk (cans of milk, type, and batch unit)</
    action_object>
<rule_conditions> deadline (date) </rule_conditions>
<remedy> late_replacement_interest (amount, 31 x7) </remedy>
</right_rule>

```

Listing 8: Replacing low-quality milk with this example.

It is assumed the rights defined in Line 1 have a name and an ID. Because the distributor has the right to revoke the right, he or she can persuade the producer that the quality of the milk was ruined during logistics due to a faulty sensor machine that was not his fault. If the distributor agrees to replace the milk, the contract's rights can be changed while it is being carried out, and the compensation can be set to false. The parties are similarly stated to be in Listing 7, and the right state is ready to be implemented immediately. Because the producer demands that the milk be replaced, the right-type is assigned conditional-right. Before the right can be exercised, the contract must be signed and the milk delivered to the producer. To replace the milk mentioned in the performance object, the performance type has been changed to cans of milk, type, and batch unit. The corresponding obligation of the distributor must be met within the 'timeframe' specified after this right is activated; otherwise, the producer is entitled to monetary compensation.

The rules for converting SLCML code to a choreography model and implementing solidity smart contract code are discussed in the following section.

5.3 Patterns and Transformation Rules

This researcher has borrowed concepts from the ADICO statements for solidity [64] and the XML for choreography model [20] translations. According to Frantz et al., [64], contract statements are broken down into various components (abbreviated as ADICO) which include; “Attributes”, “Denotic”, “AIm”, “Conditions”, and “Or-else”, where Attributes denote actor characteristics and Denotic describes obligations, permissions, or prohibitions. Conditions describe the contextual conditions of the contract, whereas Or-else describes the consequences of the action taken to regulate the contract. In addition, Frantz et al., suggests mapping rules for developers to use when creating solidity code from ADICO components. This researcher’s main contribution is to convert the SLCML rights and obligations into a choreography model based on research [20], and then into solidity code based on research [64]. The proposed SLCML-Solidity mapping is summarized in Table 7 .

This core construct mapping serves as the foundation for converting SLCML specifications into Solidity contracts. The Supply chain process choreography model is converted into a smart contract known as ‘Supply chain smart contract’(rule (a)). Other SCs can only call external functions, or be called, if the former includes interaction with other contracts. In this particular case, interactions exist between the primary smart contract, the supply chain contract, and the supply chain oracle contract, as discussed below.

Table 7: Rules for transitioning SLCML to Solidity.

Rule‘ ID	XML component	Choreography component	Solidity Code
(a)	Root element: supply chain	Supply chain: choreography model	Supply chain: smart contract
(b)	Step containing a supply chain	Choreography task	External function
(c)	Attributes	Data perspectives	Struct
(d)	Obligation	Choreography task	Function modifier, Events
(e)	Precondition	Choreography task	Function modifier
(f)	Performance type	Choreography task	Functions, Events
(g)	Right	Choreography task	Function modifier
(h)	Remedy	Embedded in model documentation	Throw statements/alternative control flow
(i)	Step containing payment	choreography task	External function

Attributes components, which are contract global variables that translate to Solidity struct members (rule (c)), are used to attach constraints such as product quantity, quality, and so on. Performance types effectively represent functions and events to reflect the mix of rights, obligations, and corresponding preconditions (rule (f)), whereas function modifiers introduce descriptive checks that invalidate function execution (rules (d, e, and g)). The performance type (rule (f)), is refined further by allowing the configuration of an item, such as an invoice, as shown in Listing 7, line 14, and a target associated with a specific operation, such as replacement, as shown in Listing 8, line 13. Events that are prompted as a result of the fulfillment of encapsulated circumstances are a subset of this type (e.g. reaching a deadline for pay). The consequences of breaking clauses in function modifiers, which are usually translated using the throw primitive (rule (h)), are referred to as remedies. A single modifier construct defines conditions linked by quantifiers, thus leaving semantic integration to the developer. For example, when a payment is made, the payment is represented as a choreography task that interacts with the merchant account. This task is implemented as an external function in Solidity (rule (i)).

Tasks in the process choreography model can also represent steps in the supply chain that interact with external resources (rule (b)). Three examples can be cited to support this: 1) data received from an external actor and passed to the smart contract; a service provider may share data regarding their service costs. 2) data about a payment task, or about configuration of transport may be sent to the smart contract by existing technology, or 3) data may be extracted from blockchain-based SCs. This external data would be handled by special contracts known as oracles. In the following section, these transformation rules for SLCML code will be used to generate the choreography model, followed by the solidity code.

5.4 SLCML to Solidity Translation

The examples of SLCML code generated in listings 1, 7, and 8, serve as the starting point for translating SLCML to Solidity. The workflow models in Figures 14 and 15 have been created using XML to choreograph transformation rules in Table 7, in which two organizations, a service consumer and a service provider, are involved in the execution of a cross-organizational milk supply chain process. A service provider organization (a milk distributor), performs a workflow process on behalf of a service consumer (a factory). However, the service provider prefers to disclose only those aspects of the workflow process that are relevant to potential customer organizations, rather than all of the details. This researcher uses BPMN notation [167] to visualize the supply chain processes specified in SLCML. Activity occurrences are differentiated using labels, with a 'c:' (for a service consumer) or a 'p:' (for a service provider) namespace marker for both the consumer's process-view request and the service provider's process-view offer.

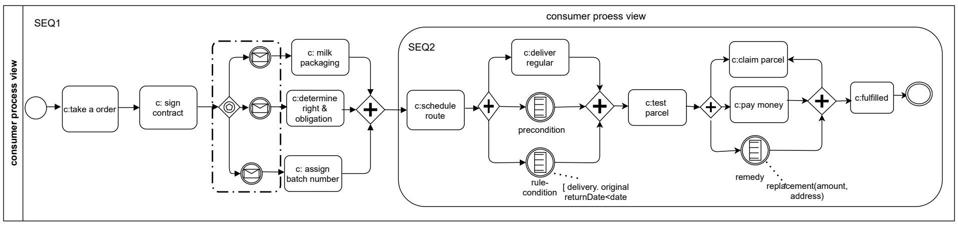


Figure 14: BPMN process view of consumer of dairy milk supply chain.

Figure 14 depicts the process flow of a factory, which begins with customer orders and ends with the sale of milk powder. Following confirmation of the agreement between the factory and the buyer (the retailer), a signed contract with an estimated delivery date is sent to the the retailer. The factory would then use its own internal process to place an order for milk packaging with a supplier, and assign a batch unit. The supplier then outsources a distribution mechanism and specifies the service provider's rights and responsibilities. To deliver the milk package, a subprocess (compound node) is executed, which includes the following tasks; the first step is to plan a route. Following that, a precondition (the quality and quantity of milk) and a rule condition are applied to the milk package before it is shipped (delivery date). The customer is eventually handed the milk package and asked to sign the receipt. If the precondition and rule condition do not match the smart contract's definitions, the customer may file a claim and request a replacement product. In contrast to the customer view in Figure 14, the private mechanism of the service provider in Figure 15, includes an event-based gateway through which the service consumer selects which component to follow throughout execution. This decision is depicted by the two message-flows in Figure 15 from the service consumer's internal domain. In contrast to the customer viewpoint, the provider process includes the activities p:determine transportation and p:determine route. During implementation, these operations must be made invisible to the service user (the customer).

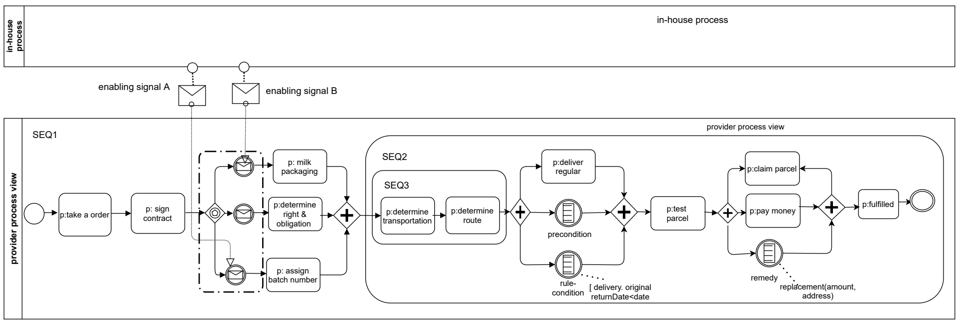


Figure 15: BPMN process for service provider.

After the process choreography model has been created, the next step is to translate it into Solidity code, as per the transformation rules shown in Table 7.

To demonstrate this process, the 'milk powder contract' as discussed above, is executed and enacted using Caterpillar [132], an open-source blockchain-based BPM system that converts business processes modeled in BPMN into SCs written in Solidity language. Listing 9 is an excerpt from the generated smart contract. To begin the task execution with rights and obligations, the smart contract, milk powder SupplyChain,' contains two events and four solidity functions based on the transformation rules. Lines 3 to 8 of Listing 9 represent global variables, and data pertaining to the process state is stored on-chain. As defined in lines 10 to 15, the list of producer and distributor variables is declared in struct, which can be accessed with a single pointer name throughout the contract. In Line 16, a further event for performance type (MilkSupply), is implemented, containing parameters such as milk quantity, producer address, and distributor address, which track the delivery of supply. Lines 17 to 20 implement the ´notifyObligationBreach event' and associated function for tracing the obligations. Similarly, an event for rights is introduced in lines 23-25 in the event that a party seeks compensation. Following that, a ´modifier precondition' is used to release the product if payment is received before the deadline.

```
1 pragma solidity ^0.4.16;
2 contract milk powder_SupplyChain{
3     uint public role;
4     address producer;
5     uint funds;
```

```
    uint milk_quantity;
    uint milk_quality;
    uint public consideration;
    .....
      struct Producer{
          address producer;
          uint role; }
     struct Distributor{
          address distributor;
          uint role;  }
event MilkSupply (uint milk_quanity, address distributor,
    address producer);
event notifyObligationBreach (*Define Type* obligaton, address
    contract );
function notify(*Define Type* obligaton, address producer){
    //TODO: Implement code to notify obligation breach for
    target contract address
    notifyObligationBreach(obligation type, contract);}
 function release(uint milk_quanity, address producer ){
     // TOD: Implement code to relase milk to the producer. }
 event claimParcel(*Define type* right, address contract);
 function replace_parcel(*Define type* right, address contract)
    {
     //TODO: Implement code to activate right for target
    contract address }
    modifier precondition(){
         //Check the condition
         uint benificiary;
         uint obligor;
         if(!paybeforedeadline){
             release(milk_quantity, producer);
             producer.send(funds); }
         else
         {  _;  } } }
```

Listing 9: Milk supply chain.

5.5 Chapter Conclusion

The goal of this chapter is to propose a pattern and transformation rules for converting SLCML code to a choreography model and then converting that model to solidity smart contract code. This researcher extends existing work that does not include transformation rules for business process views and legally binding properties to propose transformation rules that do contain legally binding properties. The SLCML contract is instantiated on the basis of the XML-based SLCML schema, implying that the language includes critical collaboration constructs as well as a framework for conceptual accuracy. A pattern and transformation rules are proposed to reduce the effort and risk associated with the development of SCs for blockchains, as well as for converting SLCML code to a choreography model, and implementing solidity smart-contract code. In addition, this researcher proposes developing a tool-assisted workflow for converting SLCML contract specifications into Solidity code. Finally, the generated solidity code of SLCML contract of the dairy supply chain running case is discussed to demonstrate the feasibility of proposed patterns and transformation rules.

6 EVALUATION OF THE SMART-LEGAL CONTRACT MARKUP LANGUAGE

6.1 Introduction

In this Chapter, the current literature is examined to identify evaluation methods for artefacts similar to those produced in this thesis. The most appropriate methods for evaluating a modeling language are chosen and tailored for the current study. An evaluation is performed to test the generality and applicability of the SLCML schema in developing legally-binding SCs.

This chapter provides a systematic evaluation of the artefacts created in this research to demonstrate their applicability in developing legally binding smart contracts. These artefacts include a legal-smart-contract ontology, a modified workflow model, and a smart-legal contract markup language (SLCML). Both ontology and workflow models are inputs for SLCML development. As a result, if the generality and applicability of the proposed language is tested, it can be assumed that the ontology, and the workflow model, can be tested as well. The generality and applicability of proposed language is tested in Chapter 5 by drafting the dairy supply chain contract (explained in Chapter 2.3.2) using proposed SLCML language. In addition, this researcher tested the generality and applicability of the language proposed in this chapter, by conducting laboratory experiments with blockchain participants in which they physically wrote contracts for the CarMan automobile running case in SLCML.

Section 6.2 describes the assessment of relevant evaluation methods. Section 6.3 evaluates the pragmatics and semantics of SLCML schema, and Section 6.4 evaluates the usability of the SLCML schema. This researcher refers the reader [20] to the complete evaluation results.

6.2 Assessment of Evaluation Methods for Modeling Languages and Their Support Tools

This section discusses existing research on evaluating modeling languages, frameworks, methodologies, and support tools for model implementation. Their benefits and drawbacks are identified and weighed in order to determine the best method for evaluating the SLCML language, furthermore, the evaluation methods selected for use in this current study are discussed, together with suggested modifications for use in this thesis.

6.2.1 Evaluation approaches:

Table 8 lists recent research relating to the evaluation and assessment of modeling languages and support tools. The *Study* column shows the article being reviewed, the *Artefact* column shows the study's research result, and the *Domain* column shows the domain to which the modeling concept is applied. The elements used to implement the modelling language are described in the *Notation* column. Finally, the *Evaluated* column tracks the assessed modelling-language aspects.

The first article, study [21], described a systematic technique for evaluating the syntax, semantic and usefulness of a modelling language. The developed evaluation technique was applied in assessing the qualitative aspects of a modelling language in innovation management in organizations. The second article, research [102], described a quantitative method for evaluating a support tool for an agent-modelling language used in software engineering. The third article, research [73], developed a modeling language for

[20] SLCML | Evaluation

customer-journey mappings in order to evaluate the support tool by measuring the correctness and utility of the models generated. The fourth article, [81], compared the results obtained using the support tool to the results obtained using the standard tool in order to evaluate a new modeling technique for software engineering. The fifth article, study [116], evaluated the significance of the methods proposed by assessing the semantic characteristics of a modeling language that extended typical agent-oriented software-engineering methodologies. The sixth article, research [131], provided a systematic method for analyzing UML semantic features. Finally, the study [47] developed a UML support tool and evaluated the consistency of models created with it, as well as the tool's utility in comparison to traditional modeling techniques.

Table 8: Evaluation-method assessments for modeling languages and support tool.

Ref.	Study Title	Artifact (ML/ST)	Domain	Notation	Evaluated		
					Syntax Corr.*	Semantic Corr.*	Usef.*
[21]	Multi-media and web-based Evaluation of Design artefacts-Syntactic, Semantic and Pragmatic Quality of Process Models	ML	Innovation management	BPMN	++	++	++
[102]	An empirical evaluation of the requirements engineering tool for socio-technical systems	ST	Software Development	AOM	+	-	+
[73]	Evaluation of modeling language for customer journeys	ST	Customer journey	CJML	+	-	+
[81]	Evaluation of the E3 Process Modeling Language Tool for the Purpose of Model creation	ST	Software development	E3 Process	-	-	+
[116]	Empirical Evaluation of Tropos4AS Modeling	ML	Software development	Tropos-4AS	-	+	-
[131]	Ontological evaluation of the UML using the Bunge-Wand-Weber model	ML	Information systems	UML	-	++	-
[47]	Evaluation of StudentUML: an Educational Tool for Consistent Modeling with UML	ST	Object oriented analysis	UML	+	-	+

* *Corr.* [Correctness]
* *Usef.* [Usefulness]

The key finding from the data presented in Table 8, was that most studies compared the utility of a modeling language's support tool to that of a traditional modeling language. Only one study [21], provided a comprehensive explanation of the evaluation approach. Furthermore, the research took into account all aspects of evaluation, such as syntactic and semantic correctness, as well as the modeling language's general applicability to the domain. Although the objective of the usability evaluation of the artefacts conducted in study [102], was similar to that of other studies, Mahunnah et al.,'s study is more relevant because the notations evaluated are similar to those evaluated in this thesis. The SLCML is a modeling language used to create SCs for deployment on a blockchain, and as a result, both studies, [21] and [102], are preferred as this thesis's evaluation method.

Table 9: Semantic qualities for assessing SLCML language, adapted from Brandtner & Helfert (2018)

Title	Description	Adaptation
Correctness	All statements in the representation	The SLCML schema correctly represents the process and elements of creating legally binding SCs.
Relevance	All statements in the representation are relevant to the problem	All of the SLCML schema elements are relevant for creating legally binding SCs.
Completeness	The representation contains all statements about the domain that are correct and relevant	The SLCML schema represents all of the elements and processes involved in creating legally binding SCs.
Authenticity	The representation gives a true account of the domain	The SLCML schema represents the elements and process of creating legally binding SCs in a realistic manner.

6.2.2 Modeling-language evaluation aspects

Brandtner et al. described a method for evaluating the syntactic, semantic and pragmatic aspects of a business-modeling framework for implementing business innovations [21]. The method has been modified and applied to blockchains as well as for the creation of SCs domains. The syntactic parts of the SLCML schema have already been specified and evaluated in [54]. This thesis only considers semantic and pragmatic aspects, with semantic quality determining how well the new SLCML schema captures contractual business features that domain experts believe are important in describing the domain. Table 9 shows the properties for measuring the semantic aspects of a new modeling language and their application to this thesis. Among these properties are the modeling language's validity, relevance to the problem domain, completeness in describing the domain, and language authenticity.

The pragmatic aspect evaluates the modeling language's perceived utility in assisting with the design, and implementation, of blockchain-enabled SCs for inter-organizational collaborations. Table 10 shows the properties for measuring the pragmatic aspects of a new modeling language and its adaptation to this thesis. Some of these characteristics are: subjective norm, image, job relevance, output quality, result demonstrability, performance, productivity, and perceived usefulness. Experts in the field of blockchain system design have taken part in evaluating both the semantic aspects and pragmatic aspects of the new modeling language.

Table 10: Pragmatic qualities for assessing SLCML language, adapted from Brandtner & Helfert (2018) [21].

Title	Explanation
Subjective Norm	People who are important would support using SLCML in creating legally binding SCs.
Image	People in my organization who use SLCML to instantiate SCs would have a high profile.
Job relevance	SLCML usage or application would be relevant in my job.
Output quality	The quality of output I get from using SLCML will be high.
Results demonstrability	I believe I could explain the benefits of using SLCML to others.
Performance	My job's performance would be improved if I used the SLCML schema.
Productivity	I would be more productive if I used the SLCML in my job.
Perceived usefulness	I find the creating of SCs through SLCML to be useful in my job.

6.2.3 Usability evaluation aspects of modeling language:

The paper [102], investigated the usability of a support tool for agent-oriented modeling language. The method of evaluation was based on a comparison of the results obtained while using the assistance language tool support with a free-hand sketch. The goal was to assess the validity of the models developed, as well as the time spent developing the models, in order to determine the benefits of the support tool.

Table 11: Properties for assessing usability of SLCML, adapted from Mahunnah et al., (2018) [102].

Item	Description
Q1	The description of the case study was clear to me.
Q2	Difficulties in modeling the legal and business requirements in SLCML.
Q3	Difficulties in choosing the business processes in SLCML.
Q4	Difficulties in modeling the use case instantiation of SLCML.
Q5	Short time is required for accomplishing the modeling SCs.
Q6	SLCML schema was very useful in the modeling of legally binding SCs.
Q7	The concepts of the SLCML schema were detailed enough to instantiate the requirements of blockchain system.

Q8	The effort of modeling SCs seems too high for the efficient use of the methodology in practice.

The properties for analyzing the performance of a modeling SLCML SCs are shown in Table 11, which was adapted from [102] for this thesis. The items assessed the difficulties, time spent, and effort expended in generating SLCML contracts. Other aspects examined include grasp of the case modeled, comprehension of the SLCML notions, and the application of SLCML in practice. The participants involved in this evaluation were newcomers to blockchain.

6.3 SLCML Schema Evaluation

A webinar event with blockchain domain experts was held to review the concepts and showcase the SLCML schema. The specialists worked in industries such as supply chain, healthcare, finance, education, and research. A total of 20 people registered for the webinar; 40% were researchers, 20% were software engineers, and 20% were data and business analysts. The balance was made up of project managers and quality assurance analysts.

All of the important SLCML schema concepts, such as business and legal aspects and relationships, were discussed during the workshop. In addition, the attendees were introduced to the concept of decentralized inter-organizational business collaboration; the main theme of this thesis. Participants provided feedback on the SLCML schema's semantic and pragmatic features at the end of the webinar.

The semantic-and-pragmatic properties of the SLCML were investigated using the modeling language evaluation approach described in Tables 9 and 10, in Section 6.2. The results of the SLCML evaluation are presented in Figure 16. The semantic characteristics are shown on the left side of the figure, and the pragmatic characteristics are depicted as perceived utility, on the right side.

On a scale of 1 to 5, blockchain domain experts agreed that the SLCML schema depicted the design process of legally binding SCs for semantic quality evaluation in an appropriate and realistic manner. They also believed that the SLCML schema accurately and comprehensively represented all of the elements required to create legally enforceable SCs. The average scores for the properties were 4.3 for realisticness, 4.3 for completeness, 4.1 for relevance, and 4.5 for accuracy. The usefulness of the SLCML schema in blockchain-related jobs, the creation of legally binding SCs output, and improved performance in SCs implementation, were all rated highly (above 4.3). The SLCML schema's increased productivity and communication, both with scores of 4.1, were two more pragmatic aspects of the SLCML schema that received high marks. The subjective norm property, which measures how important, people in the blockchain community, regard the SLCML, was also given a high score of 4.0. According to this assessment, it can be concluded that blockchain domain experts agree that the SLCML schema is semantically valid, and pragmatically beneficial, in the construction and development of legally binding SCs.

Similar scores were recorded by academic, and industry experts, in providing extra details of the semantic evaluation based on the domain experts' backgrounds. Academic, and industry experts, rated the semantic quality of the SLCML at 4.28 and 4.27, respectively. For its pragmatic aspects, domain specialists in academia and industry gave the

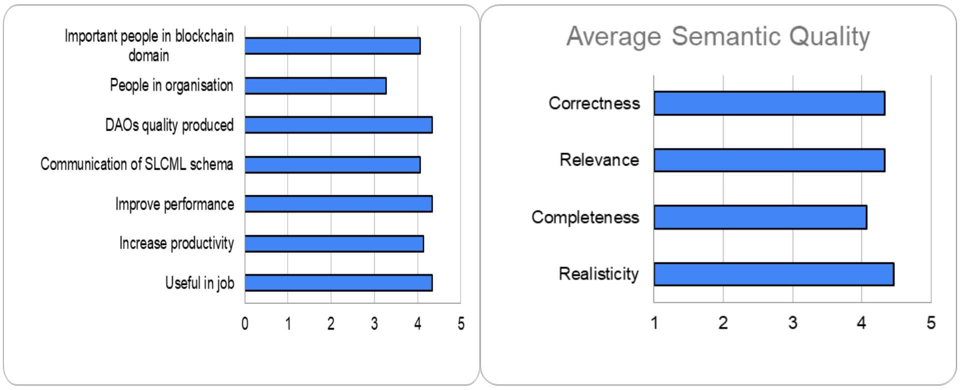


Figure 16: SLCML pragmatic- and semantic evaluation result.

SLCML similar ratings. The average pragmatic quality of the SLCML was rated 4.06 by academic experts, and 4.07 by industry professionals. These results show that academics, and industry professionals, have similar perceptions of the semantic and pragmatic features of SLCML.

6.4 SLCML Usability Evaluation

To evaluate the usability of the SLCML, a workshop was held with seven users who were not blockchain experts, but who could easily grasp the concepts of blockchain-SCs design. All seven users were master's degree students, currently working on thesis topics related to blockchain.

During the workshop session, the participants received instruction on the fundamental concepts of the SLCML schema. The participants were then divided into two groups to model a set of legally binding SCs using the SLCML schema. They were tasked with creating an example of an SC for a specific use-case in the automobile supply chain. The first group, made up of four students, worked with the proposed SLCML schema, while the second group of three students, worked with an existing smart contract modeling language (DAML). A feedback form was used to record the accuracy of the models produced, the amount of time spent, and the ease with which the assignment was completed. The participants were asked to evaluate the SLCML on a scale of 1 to 5.

The usability of developing legally binding SCs using the SLCML schema was investigated in this thesis using the modeling-language support-tool evaluation approach, as described in Table 11; Section 6.2. Usability was determined by comparing feedback data from students who developed SCs using SLCML, to students who developed SCs using the existing SC modeling language (DAML). Figure 17 depicts the results of the usability eval-uation of the SLCML. The red bars show the average scores of students who used the existing modeling language, while the blue bars show the average scores of students who used the SLCML schema.

The results in Figure 17 show that all students who attended the workshop had a similar understanding of SLCML concepts, and the running scenario. However, the amount of work required to recreate the modeling SCs, and the ease with which the SLCML could be used, varied from student to student. When compared to students who used an existing SC modeling language that required a significant amount of effort to complete the modeling task, students who used the SLCML, believed the effort required to create the legally binding SCs was quite low. Students who used the SLCML enjoyed creating legally-

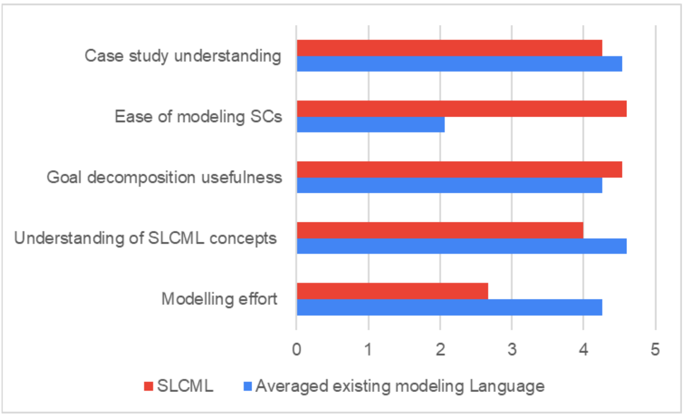


Figure 17: SLCML usability evaluation result.

binding SCs, while students who used the existing language found it difficult to model legally binding SCs. The findings show that the proposed SLCML can be used to easily design and develop legally enforceable documents.

6.5 Chapter Conclusion

The primary artefacts created during the research study are evaluated in this chapter of the thesis. One of the artefacts is the SLCML schema, which contains legal and business semantics for designing and developing legally-binding SCs. To this end, this thesis examines the literature for appropriate methodologies for evaluating the SLCML. The evaluation methodologies are compared to see how different methods evaluate the syntactic accuracy, semantic correctness, and utility of a modeling language. Based on the analysis results, appropriate SLCML evaluation techniques tailored to the current thesis are chosen. The SLCML evaluation seeks to determine the semantic quality and practical utility of the SLCML in the creation of legally binding SCs. The evaluation goal also includes determining the SLCML's effectiveness in producing SCs. The semantic and pragamtic evaluation results, as well as the usability evaluation results, are presented separately.

7 CONCLUSION

This chapter summarizes all of the research conducted in this thesis. The works related to the main chapters of the thesis are presented, and chapter summaries show how the thesis addresses each chapter's research issues. Finally, the limitations of the research presented in this thesis are discussed, together with future work that may result.

Chapter 7.1 discusses the results of Chapters 3, 4, 5 and 6, including the development of SLCML schema, semantics and syntax descriptions, solidity transformation rules, and SLCML evaluations. Section 7.2 summarizes the answers to the research questions. Section 7.4 describes the limitations and possible future work.

7.1 Discussions

The discussions emerging from this thesis are presented as follows.

7.1.1 Discussions from the development of a legally binding SCL

Chapter 3 describes the systematic development of a SCL that supports legal and business semantics, with the main contribution of the chapter being a graphical representation of future SCL development that describes legal and business semantics, and how it addresses gaps in existing approaches for building legally binding SCs. Current state-of-the-art SCLs are examined to determine whether they contain the ten properties that are required for the creation of legally-enforceable SCs. These properties, as described in Section 3.3, contribute to 'semantic suitability', 'workflow suitability' and 'expressiveness'.

Each of these categorizations are defined from a legal perspective to better understand why they are so important in making SCs legally binding. As explained in Section 3.3, semantic suitability includes the 'Who-Where-What properties' that define the parties to a contract. According to [70], a contract cannot even be entertained without having a semantically rich SCL, with the necessary vocabulary to express the 'who-where-what' of the contract. As a result, these properties become extremely important. According to [170], 'workflow suitability' properties are critical for determining whether the parties are performing their specific tasks correctly. These properties, as the name implies, outline how the contract's flow of control, data, and resources should occur, and in the event of a dispute, how a court of law can intervene to address potential problems. Finally, 'formal verifiability', or 'expressiveness', is critical for making contracts legally binding. These properties ensure that smart contracts, which are essentially computer programs, are encoded correctly based on the parties' legal intention, as defined in a natural-language contract [101, 70].

In summarizing the findings of the research questions (RQs) it was found that while most SCLs support some suitability and expressiveness properties, none of the current state-of-the-art SCLs have all ten properties listed in Table 5. While most SCLs have some semantic ('Who-concept'), and workflow properties ('control-flow' and 'data-flow'), only one SCL (Rholang), has adopted all of the 'workflow suitability' properties.

Whereas in the case of 'legally enforceable' and 'business process' SCLs, while all selected SCLs contain the three (who-where-what) 'semantic suitability' properties, they lack the syntax for interpreting the business rules, and policies that are essential factors in the formulation of rights and obligations of traditional contracts in smart contracts. In addition, current user-friendly SCLs are unable to uniquely identify who the collaborating parties are in a smart contract. Consequently, the required competencies and authorization status of the contracting parties are also not supported in the existing syntax definitions.

With respect to the ‘expressiveness’ properties, most SCLs either have ‘temporal constraints’ or ‘structural constraints’; only three SCLs, (Rholang, Lolisa, and LLL) have both properties. This is critical, because having these properties makes SCLs less prone to security vulnerabilities [161].

7.1.2 Discussions from the SCL ontology description

Chapter 4 presents the ontological concepts and properties required for the creation of legally binding SCs. This chapter builds on previous work and formalizes the legal elements of ontology in a workflow model that are critical for designing legal DAOs. HermiT reasoner software was used to to check the correctness of the proposed SCL ontology written in the web ontology language (OWL) programming language. The proposed SCL ontology and workflow model are inputs to the SLCML’s development. The properties of SCL ontology and workflow model are mapped into the previously developed eSML language (eSML 1.0). While eSML 1.0 was originally developed to incorporate the semantics of business processes in smart contracts, it lacks the legal elements which are critical to define legally-binding SCs in the context of business collaborations.

Existing SCLs, such as Solidity, Serpent, and others, are developed from an IT standpoint, in which a programmer produces machine-readable code without having any knowledge of the contract domain. Nonetheless, it should be pointed out that current research focuses on the development of SCLs in order to establish legally enforceable SCs. Researchers propose a specification language (SPESC) for collaborative design that defines the configuration of a smart contract, rather than its implementation, as cited in [75]. SPESC defines SCs code as a collaborative effort between IT specialists and domain practitioners, incorporating business or financial transactions. SPESC can be used to describe real-world contract requirements such as the party’s role, the set of terms and conditions, and so on. However, SPESC focuses on modeling legal relations between contracting parties, rather than legal positions and characteristics of contracts, such as obligation states, rights and obligations categories. In [152], researchers propose a formal specification language (Symboleo) that reflects responsibilities and powers by utilizing domain ideas and axioms. Symboleo’s standards include rights and obligations that can be tracked in real time. Formal semantics are also provided for in the form of of state charts to define the life-cycle of contracts, obligations, and authorities. While Symboleo is expressive enough to represent a wide range of real-world contracts, it lacks collaborative contract concepts and properties.

In [175], the researcher addresses the issues of formalizing natural language contracts in machine-readable languages. A contract modeling language (CML) is also proposed for modeling and expressing unstructured legal contracts that cover a wide range of common contract situations. The CML defines a natural-language comparable-clause grammar that is analogous to real-world contracts, but it does not handle transaction rules, and is unable to formalize any type of contract because it lacks domain completeness.

According to the researchers in [140], human contract intents are typically defined in natural language that is simple to understand, but is imprecise and open to interpretation. A process for developing a high-level specification that achieves common understanding through natural-language terms and is directly translated into machine instructions is also proposed. Nonetheless, this study focuses primarily on the readability and security of SCs, and does not address the domain’s collaborative contractual appropriateness, or completeness. According to [74], researchers are finding it difficult to develop SCs due to the complexity and variety of the underlying systems. Furthermore, a blockchain-independent smart-contract modeling language, called iContractML, is proposed to relieve developers of the burden of dealing with blockchains' unique complexity. The CML allows blockchain developers to focus on the business process rather than the platform's syntactic details. The CML has a very different focus and breadth than this researcher's studies. ADICO, an abbreviation for Attributes, Deontic, AIm, and Conditions, is a scala DSL that converts smart contract domain-specific components into simpler ideas [64]. A contracT tool has been developed to annotate legal-contract text using a legal-contract ontology [155], however, the proposed ontology is not yet ready to be used to create collaborative SCs. This thesis proposes a framework for the dynamic binding of parties to collaborative process roles, as well as an acceptable language for binding policy descriptions [96]. The proposed language incorporates Petri-net semantics, thus allowing policy consistency to be verified.

7.1.3 Discussions from the development of the patterns and transformation rules

The pattern and transformation rules for converting SLCML contracts to Solidity SCs are discussed in Chapter 5. Although Regnath& Steinhor [140], have proposed a method for converting conventional natural language terms into easily compiled computer language, this researcher believes that the readability and security of SCs are more important to this study than the domain's participatory contracting compatibility and completeness. As a result, study [96], proposes a methodology for interactively ratifying stakeholders for cooperative workflow functions, together with a vocabulary for enforcing policy configurations. The transformation rules from XML to a process-choreography model are proposed first, followed by the generation of Solidity code from the choreography model [20]. Nonetheless, the recommended strategy is tailored to tourist itineraries. The approach applied in this thesis is to extend the transformation rules for generating any type of domain-specific smart contract written in SLCML to Solidity code.

A number of studies related to the configuration of smart contracts have been conducted. However, an analysis of these studies show that their application to real-world agreements is limited. As a result, there is insufficient research to define collaborative and legally-enforceable SCs. This thesis bridges the gap by implementing a real-world contractual agreement concept that requires agreement from cooperating parties. This agreement written in SLCML, matches the process views from the perspective of both a service consumer and a service provider. As a result, it is clearly distinguishable from existing research that only has a purely scientific, and theoretical, focus. In addition, this researcher provides the rules for converting any SLCML smart contract in to Solidity code.

7.1.4 Discussions from the SLCML evaluations results

The evaluations conducted to assess the main artefacts produced in this thesis are detailed in Chapter 6. The SLCML schema is evaluated for its suitability for generating legally binding SCs. The semantic qualities of the SLCML schema are evaluated to determine its practicality and usefulness using the method described in [21]. The SLCML's effectiveness is determined by its usability, and ease of generating error free SCs. This is accomplished by employing the method outlined in [102].

According to the result of the SLCML schema evaluation, the modeling language is highly realistic for describing the legal, and business, elements necessary for creating legally binding SCs. According to the results of the practical (pragmatic or perceived) usefulness evaluation, the SLCML schema is quite suitable for producing high-quality legally-binding SCs. Thus, significantly improving the performance and productivity of analysts and developers in building SCs. Furthermore, the SLCML demonstrates broad applicability and utility in tasks involving the design and development of blockchain-based SCs. Some similarities and differences emerge when these findings are compared to those of a similar study [75], which demonstrates a process-based approach for modeling innovations in organizations. The results of [75] show that the modeling approach is highly relevant for tasks related to innovation management, and that the approach has a high practical usefulness. However, when compared to the other results, the average semantic qualities score is low, falling just above the second quartile (above 50%).

In comparison, the average score for both the semantic and practical usefulness of the modeling approach for the proposed SLCML schema is in the third quartile (above 75%). This demonstrates that the SLCML is not only useful for creating blockchain-based SCs, but it also accurately represents legal- and business-related elements and processes in the blockchain domain. In addition, the results of the SLCML usability evaluation show that the SLCML is highly effective and usable in producing correct SLCML contracts when compared to a freehand smart-contract modeling language. The results show that the modeling effort required to create SCs with SLCML is relatively low, whereas the effort required to create SCs with the existing modeling language is extremely high. The SLCML is simple to model and ranks in the third quartile (above 50%), while the existing modeling language (SPESC) ranks in the second quartile (below 50%). This is similar to the results of this thesis, demonstrating that SLCML is more user-friendly than an existing modeling language.

7.2 Answer to Research Questions and Chapter Summaries

The aim of this thesis is to create a SCL that can be used to generate legally-binding SCs for IOCs. The concepts of legal and business semantics, as well as blockchain decentralization, are introduced into the modeling language developed in this thesis, which is based on the eSourcing markup language (eSML 1.0). This study focuses on developing a SCL comprising the concepts and properties for drafting legally-binding collaborative business contracts.

This thesis develops 3 main research questions in order to answer this objective. The research questions are as follows: RQ1: How can a novel framework be devised for designing smart contract languages that are semantically rich, and which support the drafting of formally verifiable smart contracts for use in DAO collaboration? RQ2: How can a formal-specification language be developed for the purpose of legally-binding DAO collaboration? RQ3: How can a BPMN choreography model be built to translate an XML-based contract to blockchain-executable language? These questions are answered in the Chapters 3, 4 and 5.

The main artefacts produced in this thesis by answering the 3 research questions are: 1) The proposed framework for developing a novel legally-binding SCL, 2) The proposed SLCML; an XML-based language, and 3) The pattern and transformation rules for translating SLCML contracts to solidity. The SLCML schema is a model-driven approach for creating legally-binding SCs to address complex IOC problems. These artefacts are evaluated in Chapter 6.

This section summarizes how the main research questions are addressed in Chapters 3, 4 and 5, in that order. A summary of the evaluation of the major artefacts produced in this thesis is provided in Chapter 6.

7.2.1 Answers to RQ1

The main contribution of the first research question is to identify the suitability and expressiveness properties that are critical for developing legally-binding SCLs, and to devise a novel model for developing SCLs that include those properties. The main advantages of the proposed properties are that they are technology-independent, conceptual, and can be projected into targeted smart-contracts and blockchain technology for SCL development. The proposed suitability and expressiveness properties are based on developing legally-binding collaborative SCs. The properties of the identified SCLs were evaluated to determine whether they have the required suitability and expressiveness properties. It was deduced that most of the current state-of-the-art SCLs only partially support (not all ten properties) the suitability and expressiveness properties. By integrating those properties with state-of-the-art SCLs it will be possible to make them legally binding. The proposed framework demonstrates how a novel SCL comprising those properties can be developed, and to the best of this researcher's knowledge, this is the first framework for developing legally binding SCLs. The proposed framework can be used to improve current state-of-the-art SCLs to make them legally binding, as well as for developing novel SCLs.

7.2.2 Answers to RQ2

The main contributions of the second research question are the development of an SCL ontology, a workflow model, and a novel XML-based language SLCML. The proposed SCL ontology provides insights into the context of a smart contract, for example: who the participants to the transaction are, what they are exchanging, and under what terms and conditions this exchange takes place. The SCL ontology is evaluated using running cases drawn from existing studies, in which it outperformed standard approaches to resolving rights and obligations conflicts between parties. The proposed SCL ontology is general in nature, and can be used to configure contractual properties for a variety of other blockchain and SC applications, such as Defi ecosystems.

The workflow model provides insights into the processes and workflow patterns to be followed when conducting a smart contract from the perspective of contractual collaboration. For example: how the transactions are to be carried out, and the workflow model that will be applied. The proposed workflow model can be extended to be projected into targeted smart contracts and blockchain technology for validating a smart contract's functional correctness; thus ensuring contract security, and enabling discovery of unknown SC vulnerabilities. Also, the SCL ontology is formalized in the workflow model, to test the formal verifiability of the ontology in contractual workflow using the CPN simulation tool. Together, the proposed ontology, and workflow model, provide procedural knowledge about the expected flow of business actions within the process workflow of the individual contracting parties.

The concepts and properties of the ontology and workflow model are translated into the XML-based language SLCML; demonstrating that the SLCML contains the necessary vocabulary to define legally binding and collaborative business contracts. The generality and applicability of the SLCML is demonstrated through running cases and laboratory experiments for which SLCML code examples are provided. The evaluation results show that the SCs generated by the SLCML described, and developed, in this thesis are correct, and in accordance with legally binding contracts. Although the thesis develops the SLCML for a specific business-to-business use-case, the approach used in SLCML development could be applied to SCL development for other blockchain applications.

7.2.3 Answers to RQ3

To answer the third research question, patterns and transformation rules are proposed for translating the contract written SLCML to blockchain executable language. The patterns and transformation rules are based on ideas from existing research, and make use of the translation approach from XML to a choreography model, and from the choreography model to Solidity. The proposed patterns and transformation approach are tested by translating contracts from the dairy supply chain (introduced in Chapter 2) written in SLCML into Solidity code. As an additional test, this researcher deployed a choreography model in BPMN notation into the Caterpillar system (BPMN to Solidity translator), which generated the same Solidity code as the proposed approach. The result obtained from these tests indicate that the proposed patterns and transformation rules are a viable alternative for translating contracts written in XML-based SCLs to Solidity.

7.3 Summary of Evaluation Results

This section summarizes the results of the assessment of the main artefacts produced in this thesis. The purpose of this thesis is to conduct a systematic evaluation of the SLCML schema's utility in the creation of legally binding SCs. To accomplish this, this thesis first investigates appropriate evaluation methods for modeling approaches. Based on the findings, two appropriate evaluation methods are chosen and adapted for the artefacts produced. The first is used to evaluate the semantic, and pragmatic, qualities of the SLCML schema, while the second is used to evaluate its usability.

The semantic qualities of the SLCML are determined by how experts in the blockchain domain perceive the correctness, completeness, and relevance of the SLCML in developing legally binding SCs. Whereas, the pragmatic qualities of the SLCML are evaluated by individuals involved in blockchain-related programming tasks designed to test the usefulness, productivity, and quality of the generated SCs.

The semantic evaluation results show that the SLCML provides a realistic and correct representation of the steps involved in creating legally-binding SCs. The pragmatic evaluation results show that the SLCML is regarded as extremely useful in producing high-quality SC designs. The effectiveness of the SLCML is determined by comparing results obtained when modeling legally binding SCs with an existing SC modeling language. The results show that the SLCML requires less modeling effort, and provides greater usability in producing legally binding SCs.

7.4 Thesis Limitations and Future Work

The lack of domain completeness is the main limitation of the research presented in this thesis. the SLCML's development assistance in generating legally-binding SCs is limited to business-to-business (B2B) contracts. The use of B2B case studies in the thesis demonstrates this. Furthermore, transformation rules are proposed that are only applicable to the Solidity language. Other study limitations include the researcher's subjectivity in analyzing, and interpreting data, as well as the risk of generalization. Subjectivity related to the researcher's values and viewpoints could have an impact on the interpretation of results when analyzing the strengths and weaknesses of current approaches for building a SCL. Subjectivity may also have an impact on how the SLCML evaluation results are interpreted. The results of this thesis may be generalizable due to the large number of experts interviewed in this study. Nonetheless, blockchain technology is a relatively new innovation space, and the number of experts with the necessary knowledge to participate was limited in the interviews, webinars, and workshops held as part of this research. As a result, assembling a large number of experts to participate in several of the thesis' evaluations is a significant constraint.

This researcher intends to collaborate with business domain experts in the future to improve the SCL ontology in order to create a domain-complete ontology for smart contracting advancement. In addition, this researcher plans to use SLCML in further case studies for research projects involving the design of cyber-physical systems with smart-object orchestration. Those studies will look at expressiveness extensions in more Internet of Things (IoT) contexts. The adoption of concepts and properties that cater for advanced security assurance measures relevant for open cyber-physical system collaborations, is an important extension for the SCL ontology, and SLCML. Another unanswered research question is how to protect business collaborations using transactionality concepts other than traditional database and workflow transactions; such as side chains, tree chains, mini chains, and other blockchain-related variants. As a result, future research will look into such electronic business transactions to explore the extent SLCML ensures the security of cyber-physical system collaborations, preferably using blockchain technology. A formal analysis approach to the specification of SLCML could be developed, and this researcher plans to build a translator for the automatic conversion of SLCML instantiations into a larger set of blockchain-based languages.

List of Figures

List of Tables

References

[1] E. Abodei, A. Norta, I. Azogu, C. Udokwu, and D. Draheim. Blockchain technology for enabling transparent and traceable government collaboration in public project processes of developing economies. In *Lecture Notes in Computer Science*, pages 464–475. Springer International Publishing, 2019.

[2] E. Abodei, A. Norta, I. Azogu, C. Udokwu, and D. Draheim. Blockchain technology for enabling transparent and traceable government collaboration in public project processes of developing economies. In *Lecture Notes in Computer Science*, pages 464–475. Springer International Publishing, 2019.

[3] E. M. Al-Amaren, C. Ismail, and M. Nor. The blockchain revolution: A gamechanging in letter of credit (l/c). *International Journal of Advanced Science and Technology*, 29(3):6052–6058, 2020.

[4] S. Andrews. Introduction - learn fi, 2019. Last accessed on 30/10/2020.

[5] S. Angelov. *Foundations of B2B electronic contracting*. PhD thesis, Industrial Engineering & Innovation Sciences, 2006.

[6] T. Astigarraga, X. Chen, Y. Chen, J. Gu, R. Hull, L. Jiao, Y. Li, and P. Novotny. Empowering business-level blockchain users with a rules framework for smart contracts. In *Service-Oriented Computing*, pages 111–128, Cham, 2018. Springer International Publishing.

[7] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts (sok). In M. Maffei and M. Ryan, editors, *Principles of Security and Trust*, pages 164–186, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[8] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts (SoK). In *Lecture Notes in Computer Science*, pages 164–186. Springer Berlin Heidelberg, 2017.

[9] N. Atzei, M. Bartoletti, S. Lande, and R. Zunino. A formal model of bitcoin transactions. In S. Meiklejohn and K. Sako, editors, *Financial Cryptography and Data Security*, pages 541–560, Berlin, Heidelberg, 2018. Springer Berlin Heidelberg.

[10] M. M. Aung and Y. S. Chang. Traceability in a food supply chain: Safety and quality perspectives. *Food Control*, 39:172–184, 2014.

[11] M. Bartoletti and L. Pompianu. An empirical analysis of smart contracts: platforms, applications, and design patterns. In *International conference on financial cryptography and data security*, pages 494–509. Springer, 2017.

[12] M. Bartoletti and R. Zunino. BitML: A calculus for bitcoin smart contracts. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 83–100. Association for Computing Machinery, 2018.

[13] T. E. Beck and D. A. Plowman. Temporary, emergent interorganizational collaboration in unexpected circumstances: A study of the columbia space shuttle response effort. *Organization Science*, 25(4):1234–1252, 2014.

[14] A. Begicheva and I. Smagin. Ride: a smart contract language for waves. Technical report, 2019.

[15] K. Behnke and M. Janssen. Boundary conditions for traceability in food supply chains using blockchain technology. *International Journal of Information Management*, 52:101969, 2020.

[16] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, PLAS '16, page 91–96, New York, NY, USA, 2016. Association for Computing Machinery.

[17] A. Biryukov, D. Khovratovich, and S. Tikhomirov. Findel: Secure derivative contracts for ethereum. In *Financial Cryptography and Data Security*, pages 453–467. Springer International Publishing, 2017.

[18] S. Blackshear, E. Cheng, D. L. Dill, V. Gao, B. Maurer, T. Nowacki, A. Pott, S. Qadeer, D. Russi, S. Sezer, T. Zakian, and R. Zhou. Move : A language with programmable resources. Technical report, 2019.

[19] E. H. Boudjema, S. Verlan, L. Mokdad, and C. Faure. VYPER: Vulnerability detection in binary code. *Security and Privacy*, 3(2), Dec. 2019.

[20] A. Brahem, N. Messai, Y. Sam, S. Bhiri, T. Devogele, and W. Gaaloul. Blockchain's fame reaches the execution of personalized touristic itineraries. In *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pages 186–191, 2019.

[21] P. Brandtner and M. Helfert. Multi-media and web-based evaluation of design artifacts-syntactic, semantic and pragmatic quality of process models. *Systems, Signs and Actions: An International Journal on Information Technology, Action, Communication and Workpractices*, 11(1):54–78, 2018.

[22] M. Bravetti and G. Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In M. Lumpe and W. Vanderperren, editors, *Software Composition*, pages 34–50, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[23] M. Burg. Write your next Ethereum Contract in Pyramid Scheme, 2017. Last accessed on 30/10/2020.

[24] V. Buterin. Vyper Documentation, 2018. Last accessed on 30/10/2020.

[25] Cardano. Introduction - Cardano, 2019. Last accessed on 30/10/2020.

[26] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda. Blockchain-based traceability in agri-food supply chain management: A practical implementation. In *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany)*, pages 1–4, 2018.

[27] F. Casino, V. Kanakaris, T. K. Dasaklis, S. Moschuris, and N. P. Rachaniotis. Modeling food supply chain traceability based on blockchain technology. *IFAC-PapersOnLine*, 52(13):2728–2733, 2019. 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019.

[28] S. E. Chang, Y.-C. Chen, and M.-F. Lu. Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process. *Technological Forecasting and Social Change*, 144:1 – 11, 2019.

[29] T. Chen, K. Ding, S. Hao, G. Li, and J. Qu. Batch-based traceability for pork: A mobile solution with 2d barcode technology. *Food Control*, 107:106770, 2020.

[30] Y.-H. Chen, S.-H. Chen, and I.-C. Lin. Blockchain based smart contract for bidding system. In *2018 IEEE International Conference on Applied System Invention (ICASI)*, pages 208–211, 2018.

[31] U. W. Chohan. The decentralized autonomous organization and governance issues. *SSRN Electronic Journal*, 2017.

[32] Christian. Babbage — a mechanical smart contract language, 2017. Last accessed on 30/10/2020.

[33] C. D. Clack. Smart contract templates: Legal semantics and code validation. *Journal of Digital Banking*, 2(4):338–352, 2018.

[34] M. Coblenz, R. Oei, T. Etzel, P. Koronkevich, M. Baker, Y. Bloem, B. A. Myers, J. Sunshine, and J. Aldrich. Obsidian: Typestate and Assets for Safer Blockchain Programming. 2019.

[35] K. Crary and M. J. Sullivan. Peer-to-peer affine commitment using bitcoin. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, June 2015.

[36] S. E. S. Crawford and E. Ostrom. A grammar of institutions. *American Political Science Review*, 89(3):582–600, 1995.

[37] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer. On scaling decentralized blockchains. In J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, editors, *Financial Cryptography and Data Security*, pages 106–125, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[38] P. Dai. Smart-contract value-transfer protocols on a distributed mobile application platform. Technical report, Singapore, 2017.

[39] C. Dannen. *Introducing Ethereum and Solidity*. Apress, 2017.

[40] C. Dannen. *Introducing Ethereum and solidity*, volume 318. Springer, 2017.

[41] J. de Kruijff and H. Weigand. Ontologies for commitment-based smart contracts. In H. Panetto, C. Debruyne, W. Gaaloul, M. Papazoglou, A. Paschke, C. A. Ardagna, and R. Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, pages 383–398, Cham, 2017. Springer International Publishing.

[42] J. de Kruijff and H. Weigand. Understanding the blockchain using enterprise ontology. In E. Dubois and K. Pohl, editors, *Advanced Information Systems Engineering*, pages 29–43, Cham, 2017. Springer International Publishing.

[43] J. De Kruijff and H. Weigand. An introduction to commitment based smart contracts using ReactionRuleML. In P. E. Gordijn J., editor, *CEUR Workshop Proceedings*, volume 2239, pages 149–157. CEUR-WS, 2018.

[44] J. De Kruijff and H. Weigand. Introducing commitRuleML for smart contracts. In W. H. Johannesson P. Andersson B., editor, *CEUR Workshop Proceedings*, volume 2383, pages 1–7. CEUR-WS, 2019.

[45] E. Developers. ErgoScript, a Cryptocurrency Scripting Language Supporting Noninteractive Zero-Knowledge Proofs. Technical report, 2019.

[46] DigitalAsset. DAMLScript SDK Documentation. Technical report, 2019. Last accessed on 02/12/2020.

[47] D. Dranidis. Evaluation of studentuml: an educational tool for consistent modelling with uml. In *Proceedings of the Informatics Education Europe II Conference*, 2007.

[48] Q. DuPont. Experiments in algorithmic governance. In *Bitcoin and Beyond*, pages 157–177. Routledge, Nov. 2017.

[49] Q. DuPont. Experiments in algorithmic governance: A history and ethnography of "The DAO," a failed decentralized autonomous organization. In *Bitcoin and Beyond: Cryptocurrencies, Blockchains and Global Governance*, pages 157–177. Routledge, Nov. 2017.

[50] N. Durov. Fift : A brief introduction. Technical report, 2019.

[51] V. Dwivedi, V. Deval, A. Dixit, and A. Norta. Formal-Verification of Smart-Contract Languages: A Survey. In *Advances in Computing and Data Sciences*, pages 738–747. Springer Singapore, 2019.

[52] V. Dwivedi and A. Norta. A legal-relationship establishment in smart contracts: Ontological semantics for programming-language development. In M. Singh, V. Tyagi, P. K. Gupta, J. Flusser, T. Ören, and V. R. Sonawane, editors, *Advances in Computing and Data Sciences*, pages 660–676, Cham, 2021. Springer International Publishing.

[53] V. Dwivedi and A. Norta. Auto-generation of smart contracts from a domain-specific xml-based language. In S. C. Satapathy, P. Peer, J. Tang, V. Bhateja, and A. Ghosh, editors, *Intelligent Data Engineering and Analytics*, pages 549–564, Singapore, 2022. Springer Singapore.

[54] V. Dwivedi, A. Norta, A. Wulf, B. Leiding, S. Saxena, and C. Udokwu. A formal specification smart-contract language for legally binding decentralized autonomous organizations. *IEEE Access*, 9:76069–76082, 2021.

[55] V. Dwivedi, V. Pattanaik, V. Deval, A. Dixit, A. Norta, and D. Draheim. Legally enforceable smart-contract languages: A systematic literature review. *ACM Comput. Surv.*, 54(5), June 2021.

[56] V. K. Dwivedi and A. Norta. A legally relevant socio-technical language development for smart contracts. In *Proceedings - 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems, FAS*W 2018*, pages 11–13. Institute of Electrical and Electronics Engineers Inc., 2018.

[57] J. H. Dyer and K. Nobeoka. Creating and managing a high-performance knowledge-sharing network: the toyota case. *Strategic Management Journal*, 21(3):345–367, 2000.

[58] B. Edgington. Lll compiler documentation documentation release 0.1. Technical report, 2017.

[59] R. Eshuis, A. Norta, O. Kopp, and E. Pitkänen. Service outsourcing with process views. *IEEE Transactions on Services Computing*, 8(1):136–154, 2015.

[60] N. R. et al. Workflow resource patterns: Identification, representation and tool support. In *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, pages 216–232. Springer International Publishing, 2005.

[61] Ethereum. Solidity documentation. Technical report, 2014.

[62] S. Farrell, H. Machin, and R. Hinchliffe. Lost and found in smart contract translation—considerations in transitioning to automation in legal architecture. In *UNCITRAL, Modernizing international trade law to support innovation and sustainable development. Proceedings of the congress of the United Nations commission on international trade law*, volume 4, pages 95–104, 2017.

[63] P. D. Filippi and S. Hassan. Blockchain technology as a regulatory technology: From code is law to law is code. *First Monday*, Nov. 2016.

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

[65] W. George and C. Lesaege. A Smart Contract Oracle for Approximating Real-World, Real Number Values. In V. Danos, M. Herlihy, M. Potop-Butucaru, J. Prat, and S. Tucci-Piergiovanni, editors, *International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019)*, volume 71 of *OpenAccess Series in Informatics (OASIcs)*, pages 6:1–6:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[66] M. Giancaspro. Is a 'smart contract' really a smart idea? insights from a legal perspective. *Computer Law & Security Review*, 33(6):825–835, 2017.

[67] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1 – 101, 1987.

[68] J. Goldenfein and A. Leiter. Legal engineering on the blockchain: 'smart contracts' as legal conduct. *Law and Critique*, 29(2):141–149, May 2018.

[69] D. Golumbia. *The politics of Bitcoin: software as right-wing extremism*. U of Minnesota Press, 2016.

[70] G. Governatori, F. Idelberger, Z. Milosevic, R. Riveret, G. Sartor, and X. Xu. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artificial Intelligence and Law*, 26(4):377–409, Dec 2018.

[71] B. Gray. Conditions facilitating interorganizational collaboration. *Human Relations*, 38(10):911–936, 1985.

[72] I. Grishchenko, M. Maffei, and C. Schneidewind. Foundations and tools for the static analysis of ethereum smart contracts. In H. Chockler and G. Weissenbacher, editors, *Computer Aided Verification*, pages 51–78, Cham, 2018. Springer International Publishing.

[73] R. Halvorsrud, I. M. Haugstveit, and A. Pultier. Evaluation of a modelling language for customer journeys. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 40–48. IEEE, 2016.

[74] M. Hamdaqa, L. A. P. Metz, and I. Qasse. Icontractml: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms. In *Proceedings of the 12th System Analysis and Modelling Conference*, SAM '20, page 34–43, New York, NY, USA, 2020. Association for Computing Machinery.

[75] X. He, B. Qin, Y. Zhu, X. Chen, and Y. Liu. Spesc: A specification language for smart contracts. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 01, pages 132–137, 2018.

[76] X. He, B. Qin, Y. Zhu, X. Chen, and Y. Liu. Spesc: A specification language for smart contracts. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 01, pages 132–137, 2018.

[77] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75–105, 2004.

[78] A. home. Generate anything from any emf model, 2005.

[79] P. Howson. Building trust and equity in marine conservation and fisheries supply chain management with blockchain. *Marine Policy*, 115:103873, 2020.

[80] IOHK. plutus/extended-utxo-spec at master · input-output-hk/plutus · github, 2019. Last accessed on 30/10/2020.

[81] M. L. Jaccheri and T. Stålhane. Evaluation of the e3 process modelling language and tool for the purpose of model creation. In *International Conference on Product Focused Software Process Improvement*, pages 271–281. Springer, 2001.

[82] A. Jain, S. Arora, Y. Shukla, T. Patil, and S. Sawant-Patil. Proof of stake with casper the friendly finality gadget protocol for fair validation consensus in ethereum. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 3(3):291–298, 2018.

[83] T. Kasampalis. IELE: An Intermediate-Level Blockchain Language Designed and Implemented Using Formal Semantics. Technical report, 2018.

[84] F. A. Khalil, T. Butler, L. O'Brien, and M. Ceci. Trust in smart contracts is a process, as well. In *Financial Cryptography and Data Security*, pages 510–519. Springer International Publishing, 2017.

[85] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical Report, Ver. 2.3 EBSE Technical Report. EBSE, 2007.

[86] A. Kormiltsyn, C. Udokwu, K. Karu, K. Thangalimodzi, and A. Norta. Improving healthcare processes with smart contracts. In W. Abramowicz and R. Corchuelo, editors, *Business Information Systems*, pages 500–513, Cham, 2019. Springer International Publishing.

[87] R. M. Lee and S. D. Dewitz. Facilitating international contracting: AI extensions to edi. *International Information Systems*, 1(1):94–123, 1992.

[88] A. E. Leiponen. Competing through cooperation: The organization of standard setting in wireless telecommunications. *Management Science*, 54(11):1904–1919, 2008.

[89] X. H. Li. Blockchain-based cross-border e-business payment model. In *2021 2nd International Conference on E-Commerce and Internet Technology (ECIT)*, pages 67–73, 2021.

[90] J. Lin, Z. Shen, A. Zhang, and Y. Chai. Blockchain and iot based food traceability for smart agriculture. In *Proceedings of the 3rd International Conference on Crowd Science and Engineering*, ICCSE'18, New York, NY, USA, 2018. Association for Computing Machinery.

[91] J.-H. Lin, K. Primicerio, T. Squartini, C. Decker, and C. J. Tessone. Lightning network: a second path towards centralisation of the bitcoin economy. *New Journal of Physics*, 22(8):083022, aug 2020.

[92] R. Lin and S. Kraus. Can automated agents proficiently negotiate with humans? *Communications of the ACM*, 53(1):78–88, 2010.

[93] Z. Liu and J. Liu. Formal verification of blockchain smart contract based on colored petri net models. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 555–560, 2019.

[94] Z. Liu and J. Liu. Formal Verification of Blockchain Smart Contract Based on Colored Petri Net Models. pages 555–560. Institute of Electrical and Electronics Engineers (IEEE), jul 2019.

[95] S. Lohmann, S. Negru, and D. Bold. The protégévowl plugin: Ontology visualization for everyone. In V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, editors, *The Semantic Web: ESWC 2014 Satellite Events*, pages 395–400, Cham, 2014. Springer International Publishing.

[96] O. López-Pintado, M. Dumas, L. García-Bañuelos, and I. Weber. Dynamic role binding in blockchain-based collaborative business processes. In *Advanced Information Systems Engineering*, pages 399–414. Springer International Publishing, 2019.

[97] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 254–269, New York, NY, USA, 2016. Association for Computing Machinery.

[98] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, and A. Ponomarev. Caterpillar: A business process execution engine on the ethereum blockchain. *Software: Practice and Experience*, 49(7):1162–1193, 2019.

[99] A. Maedche and S. Staab. Ontology learning for the semantic web. *IEEE Intelligent systems*, 16(2):72–79, 2001.

[100] D. Magazzeni, P. McBurney, and W. Nash. Validation and verification of smart contracts: A research agenda. *Computer*, 50(9):50–57, 2017.

[101] D. Magazzeni, P. McBurney, and W. Nash. Validation and verification of smart contracts: A research agenda. *Computer*, 50(9):50–57, 2017.

[102] M. Mahunnah, K. Taveter, and R. Matulevičius. An empirical evaluation of the requirements engineering tool for socio-technical systems. In *2018 IEEE 7th International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 8–15. IEEE, 2018.

[103] MaiaVictor. Formality documentation release 0.3.157, Nov 2019.

[104] S. Matsuo. How formal analysis and verification add security to blockchain-based systems. In *2017 Formal Methods in Computer Aided Design (FMCAD)*, pages 1–4, 2017.

[105] R. Matulevičius, A. Norta, C. Udokwu, and R. Nõukas. Assessment of aviation security risk management for airline turnaround processes. In A. Hameurlain, J. Küng, R. Wagner, T. K. Dang, and N. Thoai, editors, *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXVI: Special Issue on Data and Security Engineering*, pages 109–141, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.

[106] A. Mavridou and A. Laszka. Designing secure ethereum smart contracts: A finite state machine based approach, 2017.

[107] A. Mavridou and A. Laszka. Tool demonstration: Fsolidm for designing secure ethereum smart contracts. In L. Bauer and R. Küsters, editors, *Principles of Security and Trust*, pages 270–277, Cham, 2018. Springer International Publishing.

[108] A. Mavridou and A. Laszka. Tool demonstration: FSolidM for designing secure ethereum smart contracts. In *Lecture Notes in Computer Science*, pages 270–277. Springer International Publishing, 2018.

[109] A. Mavridou, A. Laszka, E. Stachtiari, and A. Dubey. Verisolid: Correct-by-design smart contracts for ethereum, 2019.

[110] J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar, A. Gal, L. García-Bañuelos, G. Governatori, R. Hull, M. L. Rosa, H. Leopold, F. Leymann, J. Recker, M. Reichert, H. A. Reijers, S. Rinderle-Ma, A. Solti, M. Rosemann, S. Schulte, M. P. Singh, T. Slaats, M. Staples, B. Weber, M. Weidlich, M. Weske, X. Xu, and L. Zhu. Blockchains for business process management - challenges and opportunities. *ACM Trans. Manage. Inf. Syst.*, 9(1), Feb. 2018.

[111] J. Mendling, I. Weber, W. V. D. Aalst, J. V. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. D. Ciccio, M. Dumas, S. Dustdar, A. Gal, L. García-Bañuelos, G. Governatori, R. Hull, M. L. Rosa, H. Leopold, F. Leymann, J. Recker, M. Reichert, H. A. Reijers, S. Rinderle-Ma, A. Solti, M. Rosemann, S. Schulte, M. P. Singh, T. Slaats, M. Staples, B. Weber, M. Weidlich, M. Weske, X. Xu, and L. Zhu. Blockchains for business process management - challenges and opportunities. *ACM Trans. Manage. Inf. Syst.*, 9(1), feb 2018.

[112] L. G. Meredith, J. Pettersson, G. Stephenson, M. Stay, K. Shikama, and J. Denman. Contracts, composition, and scaling the rholang specification.

[113] A. Miller, Z. Cai, and S. Jha. Smart contracts and opportunities for formal methods. In *LNCS (including subseries LNAI and LNBI)*, volume 11247, pages 280–299. Springer Verlag, 2018.

[114] A. Miller, Z. Cai, and S. Jha. Smart contracts and opportunities for formal methods. In *International Symposium on Leveraging Applications of Formal Methods*, pages 280–299. Springer, 2018.

[115] T. Moe. Perspectives on traceability in food manufacture. *Trends in Food Science & Technology*, 9(5):211–214, 1998.

[116] M. Morandini, A. Perini, and A. Marchetto. Empirical evaluation of tropos4as modelling. *iStar*, 766:14–19, 2011.

[117] M. Möser, I. Eyal, and E. Gün Sirer. Bitcoin covenants. In J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, editors, *Financial Cryptography and Data Security*, pages 126–141, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[118] A. Mülder. Model-driven smart contract development for everyone, Sep 2019.

[119] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, 2008.

[120] N. C. Narendra, A. Norta, M. Mahunnah, L. Ma, and F. M. Maggi. Sound conflict management and resolution for virtual-enterprise collaborations. *Service Oriented Computing and Applications*, 10(3):233–251, Oct. 2015.

[121] A. Norta. Creation of smart-contracting collaborations for decentralized autonomous organizations. In R. Matulevičius and M. Dumas, editors, *Perspectives in Business Informatics Research*, pages 3–17, Cham, 2015. Springer International Publishing.

[122] A. Norta. Creation of smart-contracting collaborations for decentralized autonomous organizations. In R. Matulevičius and M. Dumas, editors, *Perspectives in Business Informatics Research*, pages 3–17, Cham, 2015. Springer International Publishing.

[123] A. Norta. Establishing distributed governance infrastructures for enacting cross-organization collaborations. In A. Norta, W. Gaaloul, G. R. Gangadharan, and H. K. Dam, editors, *Service-Oriented Computing – ICSOC 2015 Workshops*, pages 24–35, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

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

[125] A. Norta and R. Eshuis. Specification and verification of harmonized business-process collaborations. *Information Systems Frontiers*, 12(4):457–479, Apr. 2009.

[126] A. Norta and P. Grefen. Discovering patterns for inter-organizational business process collaboration. *International Journal of Cooperative Information Systems*, 16(03n04):507–544, 2007.

[127] A. Norta and L. Kutvonen. A cloud hub for brokering business processes as a service: A "rendezvous" platform that supports semi-automated background checked partner discovery for cross-enterprise collaboration. In *2012 Annual SRII Global Conference*, pages 293–302, 2012.

[128] A. Norta, L. Ma, Y. Duan, A. Rull, M. Kõlvart, and K. Taveter. eContractual choreography-language properties towards cross-organizational business collaboration. *Journal of Internet Services and Applications*, 6(1):1–23, Apr. 2015.

[129] A. Norta, A. B. Othman, and K. Taveter. Conflict-resolution lifecycles for governed decentralized autonomous organization collaboration. In *Proceedings of the 2015 2nd International Conference on Electronic Governance and Open Society: Challenges in Eurasia*, EGOSE '15, page 244–257, New York, NY, USA, 2015. Association for Computing Machinery.

[130] OCamlPro. Welcome to Liquidity's documentation! — Liquidity 1.05, 2018. Last accessed on 30/10/2020.

[131] A. L. Opdahl and B. Henderson-Sellers. Ontological evaluation of the uml using the bunge–wand–weber model. *Software and systems modeling*, 1(1):43–67, 2002.

[132] OrlenysIp. orlenyslp/caterpillar, 2019.

[133] Q. Pan and X. Koutsoukos. Building a blockchain simulation using the Idris programming language. In *ACMSE 2019 - Proceedings of the 2019 ACM Southeast Conference*, pages 190–193. Association for Computing Machinery, Inc, 2019.

[134] M. P. Papazoglou. Service-oriented computing: concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering, 2003. WISE 2003.*, pages 3–12, 2003.

[135] T. Patron. *The Bitcoin Revolution: An Internet of Money*. Travis Patron, 2015.

[136] M. Pilkington. Blockchain technology: principles and applications.

[137] S. Popejoy. The pact smart contract language. *June-2017.[Online]. Available: http://kadena. io/docs/Kadena-PactWhitepaper. pdf*, 2016.

[138] S. Pourmirza, S. Peters, R. Dijkman, and P. Grefen. Bpms-ra: A novel reference architecture for business process management systems. *ACM Trans. Internet Technol.*, 19(1), feb 2019.

[139] W. W. Powell, K. W. Koput, and L. Smith-Doerr. Inter organizational collaboration and the locus of innovation: Networks of learning in biotechnology. *Administrative Science Quarterly*, 41(1):116–145, 2021/06/21/ 1996.

[140] E. Regnath and S. Steinhorst. Smaconat: Smart contracts in natural language. In *2018 Forum on Specification Design Languages (FDL)*, pages 5–16, 2018.

[141] R. Revere. functional-solidity-language, 2017. Last accessed on 30/10/2020.

[142] D. Robinson. Ivy : A declarative predicate language for smart contracts introduction : Two blockchain models, nov 2017.

[143] T. Ruokolainen, S. Ruohomaa, and L. Kutvonen. Solving service ecosystem governance. In *2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops*, pages 18–25, 2011.

[144] N. Russell, A. H. Ter Hofstede, W. M. Van Der Aalst, and N. Mulyar. Workflow control-flow patterns: A revised view. *BPM Center Report BPM-06-22, BPMcenter. org*, pages 06–22, 2006.

[145] N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow data patterns: Identification, representation and tool support. pages 353–368, 2005.

[146] N. Russell, W. M. Van Der Aalst, and A. H. Ter Hofstede. *Workflow patterns: the definitive guide*. MIT Press, 2016.

[147] A. Savelyev. Contract law 2.0: 'smart' contracts as the beginning of the end of classic contract law. *Information & Communications Technology Law*, 26(2):116–134, Apr. 2017.

[148] F. Schrans, S. Eisenbach, and S. Drossopoulou. Writing safe smart contracts in Flint. In *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming - Programming'18 Companion*, pages 218–219, New York, New York, USA, 2018. ACM Press.

[149] D. Seidl and F. Werle. Inter-organizational sensemaking in the face of strategic meta-problems: Requisite variety and dynamics of participation. *Strategic Management Journal*, 39(3):830–858, 2018.

[150] P. L. Seijas and S. Thompson. Marlowe: Financial contracts on blockchain. In *Lecture Notes in Computer Science*, pages 356–375. Springer International Publishing, 2018.

[151] I. Sergey, V. Nagaraj, J. Johannsen, A. Kumar, A. Trunov, and K. C. G. Hao. Safer smart contract programming with scilla. *Proc. ACM Program. Lang.*, 3(OOPSLA), Oct. 2019.

[152] S. Sharifi, A. Parvizimosaed, D. Amyot, L. Logrippo, and J. Mylopoulos. Symboleo: Towards a specification language for legal contracts. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*, pages 364–369, 2020.

[153] S. Shekhar and H. Xiong. Agent-based models. In *Encyclopedia of GIS*, pages 11–11. Springer US, 2008.

[154] A. Smajgl, L. R. Izquierdo, and M. Huigen. Rules, knowledge and complexity: How agents shape their institutional environment. *Journal of Modelling & Simulation of Systems*, 1(2), 2010.

[155] M. Soavi, N. Zeni, J. Mylopoulos, and L. Mich. ContracT – from legal contracts to formal specifications: Preliminary results. In *Lecture Notes in Business Information Processing*, pages 124–137. Springer International Publishing, 2020.

[156] S. Suriadi, R. Andrews, A. H. M. Ter Hofstede, and M. T. Wynn. Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. *Information Systems*, 64:132 – 150, 2017.

[157] M. Swan. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[158] H. Syahputra and H. Weigand. The development of smart contracts for heterogeneous blockchains. In K. Popplewell, K.-D. Thoben, T. Knothe, and R. Poler, editors, *Enterprise Interoperability VIII*, pages 229–238, Cham, 2019. Springer International Publishing.

[159] N. Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), Sep. 1997.

[160] N. Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), Sept. 1997.

[161] T. Tateishi, S. Yoshihama, N. Sato, and S. Saito. Automatic smart contract generation using controlled natural language and template. *IBM Journal of Research and Development*, 63(2/3):6:1–6:12, mar 2019.

[162] Tezos. Michelson : the language of Smart Contracts in I - Semantics. 2018. Last accessed on 30/10/2020.

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

[164] N. Valliappan, S. Mirliaz, E. Lobo Vesga, and A. Russo. Towards adding variety to simplicity. In T. Margaria and B. Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*, pages 414–431, Cham, 2018. Springer International Publishing.

[165] N. Valliappan, S. Mirliaz, E. L. Vesga, and A. Russo. Towards adding variety to simplicity. In *Lecture Notes in Computer Science*, pages 414–431. Springer International Publishing, 2018.

[166] R. H. Von Alan, S. T. March, J. Park, and S. Ram. Design science in information systems research. *MIS quarterly*, 28(1):75–105, 2004.

[167] M. von Rosing, S. White, F. Cummins, and H. de Man. Business process model and notation-bpmn., 2015.

[168] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu, and J. Kishigami. Blockchain contract: Securing a blockchain applied to smart contracts. In *2016 IEEE International Conference on Consumer Electronics (ICCE)*, pages 467–468, 2016.

[169] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling. Untrusted business process monitoring and execution using blockchain. In *Lecture Notes in Computer Science*, pages 329–347. Springer International Publishing, 2016.

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

[171] T. Weingaertner, R. Rao, J. Ettlin, P. Suter, and P. Dublanc. Smart contracts using blockly: Representing a purchase agreement using a graphical programming language. In *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 55–64, 2018.

[172] J. Wilcke. Mutan Language, 2015. Last accessed on 30/10/2020.

[173] A. Windeler and J. Sydow. Project networks and changing industry practices collaborative content production in the german television industry. *Organization Studies*, 22(6):1035–1060, 2001.

[174] A. Workgroups. Sophia introduction, Nov 2017.

[175] M. Wöhrer and U. Zdun. Domain specific language for smart contract development. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9, 2020.

[176] B. Xiao, X. Fan, S. Gao, and W. Cai. Edgetoll: A blockchain-based toll collection system for public sharing of heterogeneous edges. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2019.

[177] Z. Yang and H. Lei. Lolisa: Formal syntax and semantics for a subset of the solidity programming language, 2018.

[178] Yao-Hua Tan and W. Thoen. Modeling directed obligations and permissions in trade contracts. In *Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, volume 5, pages 166–175 vol.5, 1998.

[179] H. Yoichi. Morphing smart contracts with bamboo, Nov 2017.

[180] J. Zhu, S. Xu, I. Weber, A. B. Tran, P. Rimba, A. Ponomarev, S. Falamaki, S. Chen, and M. Staples. Risks and opportunities for systems using blockchain and smart contracts. 2017.

Acknowledgements

I would like to thank my thesis supervisor, Alex Norta, for his patience and guidance throughout the process of writing this thesis and conducting the research studies that resulted in this thesis.

I'd also like to thank Dirk Draheim, who started this journey with me as a co-supervisor and helped to lay the groundwork for SLR work.

I acknowledge Alexander Wulf's contribution to the creation of the smart contract law ontology through his legal expertise.

I would like to thank the faculty and departmental staff, particularly Katri Kadakas and Siiri Taveter, for their assistance during my PhD research.

I'd also like to thank the QTUM foundation for helping to fund some of my PhD research.

I'd like to thank my friend Vishwajeet Patanaik for his inspiration and advice.

Thank you, Doug Robinson, for being very thorough and taking time to fixmy English articles and commas. I learned a lot in the process and this knowledge will help my future scientific publications to comply with US or UK grammar rules.

I thank my wife Sweksha shukla, for endless encouragement and keeping me on track, when I was loosing hope that I will ever finish this thesis.

And last but not least—thanks goes to my parents, who has throughout my life encouraged me to study, study, study..

Abstract
A Legally-Relevant Socio-Technical Language Development for Smart Contracts

Inter-organizational collaborations (IOCs) are critical for any organization that relies on other organizations to perform functions that are outside of its core competencies. Trust issues such as security, interoperability, and transparency exist in systems and processes that support organizational collaborations. Due to security concerns, the party in charge of centrally controlled collaboration system can modify and tamper the exchange of information among participating stakeholders. Furthermore, access control cannot be applied to information assets that have been exchanged. With lacking interoperability, data availability between collaborating parties cannot be guaranteed in real time. Thus, there is a lack of transparency and it is difficult to detect unauthorized manipulations by the centralized entity in charge of the collaboration. Decentralized governance of inter-organizational collaborations has been proposed as a solution for mitigating issues in trust-management research. Integrating these governance systems with traditional information systems is difficult, if not, impossible.

By enabling the execution, monitoring, and improvement of business processes within, or across business networks, emerging blockchain technology has the potential to transform the business ecosystem. Blockchain is a peer-to-peer network that allows for decentralized data storage by replicating data across the network's participants' nodes. Even in networks where nodes do not trust one another, blockchains enable processes to be carried out in a secure environment. Peer-to-peer networks, consensus mechanisms, cryptography techniques, and a distributed ledger are all used to aid in this process. For business processes, smart contracts (SCs) are a critical concept in the blockchain ecosystem. Inter-organizational business processes and governance systems, can be coded into computer-executable programs using SCs to ensure automated accountability of those involved in organizational collaborations. Blockchain technology governs inter-organizational business processes and, through SCs, allows for decentralized autonomous organizations (DAO) with governance capabilities.

A DAO demonstrates how blockchain technology can be used to improve efficiency, effectiveness, and quality while automating business collaboration. Unfortunately, despite the fact that blockchains are designed to provide a technological framework for drafting DAO SCs that are legally binding, the underlying contractual concepts and properties required are still less researched to make said SCs legally binding. Furthermore, the well-known Ethereum crowdfunding DAO attack is an illustration of how a misalignment between the semantics of SCLs and the intuition of programmers result in massive financial losses.

Established SCLs, such as Solidity, Serpent, and others, are developed from an IT standpoint. Due to the programmer's lack of prior knowledge of the contract domain, SCs written in these languages are ambiguous and full of bugs. Still, several workarounds including SPESC, Symboleo, and SmaCoNat for developing SCLs currently exist that support the legally binding concepts and properties of SCs. Despite the fact that the aforementioned SCLs present intriguing approaches for the development of legally binding SCs, most (if not all) of these solutions either lack domain-completeness, or are intended for non-collaborative business processes.

This thesis addresses the existing shortcomings by developing an ontology that incorporates legally binding- and collaborative contractual- concepts and properties. The SCL ontology is mapped into Colored Petri Nets (CPNs) that can be used to design, develop,

and analyze the processing state of SCs in order to track the fulfillment of contractual properties. The concepts and properties captured in the SCL ontology is translated into the XML-based language SLCML (smart-legal-contract markup language), in which blockchain developers focus on the contractual workflow rather than the syntax specifics of each blockchain platform. To reduce the effort and risk associated with smart-contract development, this study proposes a set of transformation rules for caterpillar to automatically generate SLCML contracts in Solidity. This thesis provides solutions for stakeholders to understand SCs and incorporate contractual business-process semantics. Furthermore, the proposed approach reduces errors caused by conceptual differences between the contractual clause and the applicable code.

The development of artefacts in this thesis adheres to the design-science research method, which entails the rigorous creation, and evaluation of artefacts. Two evaluation methodologies, running cases (use cases), and lab experiments, are used to demonstrate the generality and applicability of artefacts in this thesis. Two running cases namely, the automotive, and the dairy supply chains, are developed to evaluate this work. During lab experiments, SLCML is examined to determine its generality and applicability, such as semantic qualities, and pragmatic usefulness. The semantic qualities validate the SLCML's ability to generate realistic, complete, relevant, and correct SCs for IOCs. The pragmatic properties assess the SLCML's practical usefulness in blockchain-related development tasks, such as increasing the productivity and output quality for business-to-business SC development. This study compares the designing effort and ease of creating SCs with SLCML, and existing modeling languages such as SPESC, DAML, and others to determine the effectiveness of SLCML in specifying correct and complete legally binding DAOs. The evaluation results show that the SLCML's outputs have a high degree of utility and correct representations of legally binding SCs. The evaluation of the SLCML reveals that creating SCs requires very little effort and is intuitive to perform.

Kokkuvõte
Arukate lepingute jaoks õiguslikult asjakohane sotsiaal-tehniline keelearendus

Organisatsioonidevaheline koostöö (IOC) on kriitilise tähtsusega igale organisatsioonile, mis tugineb muudele organisatsioonidele funktsioonide täitmisel, mis jäävad väljapoole tema põhipädevusi. Usaldusprobleemid, nagu turvalisus, koostalitlusvõime ja läbipaistvus, eksisteerivad süsteemides ja protsessides, mis toetavad organisatsioonilist koostööd. Turvaprobleemide tõttu võib koostöösüsteemi eest vastutav osapool teabevahetust kokkumängu teinud sidusrühmade vahel muuta ja rikkuda. Lisaks ei saa vahetatud teabevaradele juurdepääsu kontrolli rakendada. Koostalitlusvõime puudumise tõttu ei saa tagada andmete kättesaadavust vandenõu osapoolte vahel reaalajas. Läbipaistvuse puudumise tõttu on koostöö eest vastutava tsentraliseeritud üksuse volitamata manipuleerimist raske tuvastada. Usaldusjuhtimise uuringute probleemide leevendamise lahendusena on pakutud välja organisatsioonidevahelise koostöö detsentraliseeritud valitsemine. Nende juhtimissüsteemide integreerimine traditsiooniliste infosüsteemidega on keeruline, kui mitte, siis võimatu.

Võimaldades äriprotsesside elluviimist, jälgimist ja täiustamist ärivõrkudes või nende vahel, võib arenev plokiahela tehnoloogia muuta äri ökosüsteemi. Blockchain on peer-to-peer võrk, mis võimaldab detsentraliseeritud andmete salvestamist, kopeerides andmeid võrgus osalejate sõlmedes. Isegi võrkudes, kus sõlmed üksteist ei usalda, võimaldavad plokiahelad protsesse läbi viia turvalises keskkonnas. Selle protsessi abistamiseks kasutatakse peer-to-peer võrke, konsensusmehhanisme, krüptograafiatehnikaid ja hajutatud pearaamatut. Äriprotsesside jaoks on nutikad lepingud (SC) plokiahela ökosüsteemis kriitilise tähtsusega. Organisatsioonidevahelised äriprotsessid ja juhtimissüsteemid saab kodeerida arvutiga käivitatavateks programmideks, kasutades SC-sid, et tagada organisatsioonilises koostöös osalejate automatiseeritud vastutus. Plokiahela tehnoloogia juhib organisatsioonidevahelisi äriprotsesse ja võimaldab SC-de kaudu detsentraliseeritud autonoomseid organisatsioone (DAO).

DAO näitab, kuidas plokiahela tehnoloogiat saab kasutada tõhususe, tulemuslikkuse ja kvaliteedi parandamiseks, automatiseerides samal ajal ärikoostööd. Vaatamata asjaolule, et plokiahelad on loodud pakkuma tehnoloogilist raamistikku DAO SC-de koostamiseks, mis on õiguslikult siduvad, on paraku nõutavad lepingulised kontseptsioonid ja omadused siiski vähem uuritud, et muuta need SC-d õiguslikult siduvaks. Samuti on DAO rünnak näide sellest, kuidas SCL-ide semantika ja programmeerija intuitsiooni vaheline lahknevus võib põhjustada suuri rahalisi kaotusi. Konfliktide (rünnakute) korral on SC-käitamise varasemate toimingute jälgimine keeruline ja kulukas, kuna SC-d ei tea oma töötlemisolekust.

Väljakujunenud SCL-id, nagu Solidity, Serpent ja teised, on välja töötatud IT seisukohast. Kuna programmeerijal puuduvad lepingudomeeni eelteadmised, on neis keeltes kirjutatud SC-d mitmetähenduslikud ja täis vigu. Siiski on praegu olemas mitmeid lahendusi, sealhulgas SPESC, Symboleo ja SmaCoNat SCL-ide arendamiseks, mis toetavad SC-de õiguslikult siduvaid kontseptsioone ja omadusi. Hoolimata asjaolust, et ülalnimetatud SCL-id pakuvad intrigeerivaid lähenemisviise õiguslikult siduvate SC-de arendamiseks, puudub enamikul (kui mitte kõigil) neist lahendustest domeeni täielikkus või need on mõeldud mittekoostöölistele äriprotsessidele.

See lõputöö käsitleb olemasolevaid puudusi, töötades välja ontoloogia, mis hõlmab õiguslikult siduvaid ja koostöölepingulisi mõisteid ja omadusi. SCL ontoloogia on vormistatud värvilistes Petri võrkudes (CPN), mida saab kasutada SC-de töötlusoleku kavandamiseks, arendamiseks ja analüüsimiseks, et jälgida lepinguliste omaduste täitmist. SCL-i ontoloogiasse jäädvustatud mõisted ja omadused tõlgitakse XML-põhisesse keelde SLCML (smart-legal-contract markup language), milles plokiahela arendajad keskenduvad pigem lepingulisele töövoole kui iga plokiahela platvormi süntaksi spetsiifikale. Nutika lepingute arendamisega seotud pingutuste ja riskide vähendamiseks pakub see uuring välja Caterpillari jaoks ümberkujundamise reeglid, mis loovad Solidity'is automaatselt SLCML-lepingud. See lõputöö pakub sidusrühmadele lahendusi SC-de mõistmiseks ja lepinguliste äriprotsesside semantika kaasamiseks. Lisaks vähendab pakutud lähenemisviis lepinguklausli ja kohaldatava koodi kontseptuaalsetest erinevustest põhjustatud vigu.

Artefaktide väljatöötamisel käesolevas lõputöös järgitakse disain-teaduslikku uurimismeetodit, mis eeldab esemete ranget loomist ja hindamist. Artefaktide üldistuse ja rakendatavuse demonstreerimiseks selles lõputöös kasutatakse kahte hindamismetoodikat: jooksvaid juhtumeid (kasutusjuhtumeid) ja laborikatseid. Selle töö hindamiseks töötatakse välja kaks jooksvat juhtumit, nimelt autotööstus ja piimatoodete tarneahelad. Laboratoorsete katsete käigus uuritakse SLCML-i, et teha kindlaks selle üldistus ja rakendatavus, näiteks semantilised omadused ja pragmaatiline kasulikkus. Semantilised omadused kinnitavad SLCML-i võimet luua IOC-de jaoks realistlikke, täielikke, asjakohaseid ja õigeid nutikaid lepinguid (SC). Pragmaatilised omadused hindavad SLCML-i praktilist kasulikkust plokiahelaga seotud arendusülesannetes, nagu tootlikkuse ja toodangu kvaliteedi tõstmine, ettevõtetevaheliste SC-de arendamisel. Selles uuringus võrreldakse SLCML-i ja olemasolevate modelleerimiskeelte (nt SPESC, DAML ja teiste) projekteerimistööd ja SC-de loomise lihtsust, et teha kindlaks SLCML-i tõhusus õigete ja täielike juriidiliselt siduvate DAO-de määramisel. Hindamistulemused näitavad, et SLCML-i väljundid on väga realistlikud ja korrektsed õiguslikult siduvate SC-de esitused. SLCML on äärmiselt kasulik DAO projekteerimisülesannete täitmisel toodetud plokiahela DAO-de jõudluse ja kvaliteedi suurendamiseks. SLCML-i hindamine näitab, et SC-de loomine nõuab väga vähe pingutusi ja seda on äärmiselt lihtne kasutada.

Appendix 1

I

V. Dwivedi, V. Pattanaik, V. Deval, A. Dixit, A. Norta, and D. Draheim. Legally enforceable smart-contract languages: A systematic literature review. *ACM Comput. Surv.*, 54(5), June 2021

Legally Enforceable Smart-Contract Languages: A Systematic Literature Review

VIMAL DWIVEDI, Blockchain Technology Group, Tallinn University of Technology, Estonia
VISHWAJEET PATTANAIK, Information Systems Group, Tallinn University of Technology, Estonia
VIPIN DEVAL, ABHISHEK DIXIT, and ALEX NORTA, Blockchain Technology Group, Tallinn University of Technology, Estonia
DIRK DRAHEIM, Information Systems Group, Tallinn University of Technology, Estonia

Smart contracts are a key component of today's blockchains. They are critical in controlling decentralized autonomous organizations (DAO). However, smart contracts are not yet legally binding nor enforceable; this makes it difficult for businesses to adopt the DAO paradigm. Therefore, this study reviews existing Smart Contract Languages (SCL) and identifies properties that are critical to any future SCL for drafting legally binding contracts. This is achieved by conducting a Systematic Literature Review (SLR) of white- and grey literature published between 2015 and 2019. Using the SLR methodology, 45 Selected and 28 Supporting Studies detailing 45 state-of-the-art SCLs are selected. Finally, 10 SCL properties that enable legally compliant DAOs are discovered, and specifications for developing SCLs are explored.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Software and its engineering** → **Context specific languages**; • **Networks** → *Network protocols*; • **Computer systems organization** → Peer-to-peer architectures;

Additional Key Words and Phrases: Blockchain, decentralized autonomous organization, expressiveness, smart contract language, suitability, systematic literature review

ACM Reference format:
Vimal Dwivedi, Vishwajeet Pattanaik, Vipin Deval, Abhishek Dixit, Alex Norta, and Dirk Draheim. 2020. Legally Enforceable Smart-Contract Languages: A Systematic Literature Review. *ACM Comput. Surv.* 54, 5, Article 110 (May 2021), 34 pages.
https://doi.org/10.1145/3453475

1 INTRODUCTION

Introduced nearly a decade ago, blockchains have been the underlying data structure and enabling technology for supporting distributed data applications across a plethora of sociotechnical sys-

This research work is partially funded by the Qtum Foundation, Singapore.
Authors' addresses: V. Dwivedi (corresponding author), V. Deval, A. Dixit, and A. Norta, Blockchain Technology Group, Tallinn University of Technology, Akadeemia tee 15/2, Tallinn, Harju County, Estonia, 12618; emails: {vimal.dwivedi, vipin.deval, abhishke.dixit}@taltech.ee, alex.norta.phd@ieee.org; V. Pattanaik and D. Draheim, Information Systems Group, Tallinn University of Technology, Akadeemia tee 15a, Tallinn, Harju County, Estonia, 12616; emails: {vishwajeet.pattanaik, dirk.draheim}@taltech.ee.

0360-0300/2020/05-ART110 $15.00
https://doi.org/10.1145/3453475

tems from both academia and industry. Blockchain technologies realize immutable, trusted, and decentralized data-storage systems by combining a series of innovations from the areas of distributed computing and cybersecurity [75]. Thus, blockchain technologies are applied in a wide variety of domains such as e-commerce [32], e-health [10, 111], e-governance [1, 2, 35, 62], and finance [17], among others. The inherent features of blockchains that have enabled its application in so many domains, has also made it possible to achieve Nick Szabo's vision of smart contracts that he defines as "a set of promises, specified in a digital form, including protocols within which the parties perform on these promises" [97]. Enabling the capture, verification, validation and enforcement of terms agreed upon by multiple parties, blockchain technologies allow for the implementation of trustless- and transparent smart contracts. This is possible because blockchain-based smart-contract transactions are stored on encrypted, distributed ledgers, and parties can carry out transactions and agreements anonymously, without the need for central entities, or external legal-enforcement systems.

Based on recent literature, there has been a rising interest in the development of **Decentralized Autonomous Organizations (DAO)**. A DAO is an organization (or a consortium) "that is run through rules encoded as computer programs called *smart contracts*" [23]. DAO smart contracts are computer programs that run on peer-to-peer networks, incorporating governance and decision-making rules [95]. Unfortunately, although blockchains comprise the technological framework by design that potentially facilitates drafting DAO smart contracts supported by the law. Still, the underlying contractual concepts and properties necessary to render said smart contracts legally binding (which we refer to as "suitability" [77]), are still less researched [23, 45, 47]. As Norta et al. explain, ontological *suitability* of DAO **smart-contract languages (SCL)s** can be realized as (i) the choreography or workflow of processes [18] in DAOs (viz. *concepts*) and (ii) the semantics that define the individual DAO processes (viz. *properties*) [77].

Additionally, although several novel SCLs such as Solidity,[1] Michelson,[2] and Rholang[3] have been developed for implementing executable smart contracts, unfortunately, many of said SCLs are often error-prone [65] and hard to debug due to the blockchains' immutability [60]. An infamous example of how the lack of formally verifiable smart contracts could potentially lead to massive financial losses is the 2016 attack on "The DAO" (the first implementation of crowd-funding focused DAO) [57] where hackers were able to steal 50 million dollars from the DAO due to "call to the unknown" and "re-entrancy" vulnerabilities [36]. Such incidents have led to the development of several formal verification techniques (e.g., Reference [52]) that are used to validate the correctness of intended behavior of smart contracts [14]. Drawing influence from previous work [39], in this article, we refer to these formal verification techniques as expressiveness.

Given that suitability- and *expressiveness* SCL-properties are sine qua non for drafting and enforcing formally verifiable, legally binding smart contracts in DAO collaborations; through this **Systematic Literature Review (SLR)**, we aim to conduct a comprehensive meta-study of existing SCLs developed with a focus on the coding of smart contracts for DAOs. To achieve this, we follow the Kitchenham's guidelines for conducting SLR in software engineering [59] and therefore, conduct the study using a five-step process. Based on our specific **research questions (RQ)s**, we first identify relevant keywords (and their synonyms) typically used in the scientific literature. By conducting a thorough search of various academic databases and technology-related online publishing platforms, we identify 616 articles published as white- and grey literature. We then

[1]Solidity, docs page.
[2]Tezos home page.
[3]Rholang GitHub page.

examine the titles, keywords, and abstracts of these articles to extract relevant articles that propose (or discuss) novel SCLs, using a predefined set of inclusion- and exclusion criteria. The selected 130 primary studies are then comprehensively studied and scored on the basis of a two-stage quality-assessment phase. The 73 articles that explicitly answer the research questions *RQ1* and *RQ2* below, are identified and indicated as selected studies and supporting studies. Thereafter, we examine the 45 SCLs proposed in the identified selected studies and segregate them based on their attributes and applications. By examining the remaining 28 supporting studies, we identify the available properties and propose new ones that are critical when developing legally binding SCLs. Finally, to assess if the identified SCLs have the required suitability and expressiveness properties necessary for developing DAO smart contracts, we also exhaustively scrutinize the selected supporting studies and their references. Next, we deduce that most of the state-of-the-art SCLs only partially support the business' contractual processes. Additionally, based on the findings of the SLR, we devise a novel model for designing SCLs that are semantically rich and support the drafting of formally verifiable smart contracts, aimed towards DAO collaborations.

In particular, we attempt to answer the following research questions:

- *RQ1*: What blockchain-based SCLs already exist in scientific- and non-scientific literature?
- *RQ2*: What properties of business-oriented SCLs constitute to suitability and expressiveness?
- *RQ3*: What are the obstacles in existing SCLs that restrict the achievement of business-contractual objectives?

The remaining article is structured as follows. Section 2 provides additional information relevant to understanding the functionality of blockchain technology and smart contracts. This section also provides insights about previously published studies and surveys that discuss state-of-the-art SCLs. Section 3 details the methodology used for conducting this SLR. In Section 4, we elucidate 45 state-of-the-art SCLs identified after the quality-assessment phase of the SLR. Section 5 and Section 6 answer the research questions RQ2 and RQ3 and enumerate the properties that enable suitability and expressiveness in SCLs. In Section 7, we discuss the pitfalls of the available SCLs and thereby introduce a novel model for designing legally enforceable SCLs. Finally, Section 8 reviews the threats to the validity of this SLR, and Section 9 concludes this article by summarizing our findings and briefly discussing the open issues of our future work.

2 BACKGROUND AND RELATED WORK

To illustrate the legal implications of blockchain smart-contracts, we first present the basics of blockchains and then discuss the benefits of smart contracts. We then present the technical overview of SCLs in Section 2.1 and further briefly discuss the legal implication of smart contracts in Section 2.2.

2.1 Blockchains and Smart Contracts

Blockchain technology has been introduced in 2009 with the cryptocurrency Bitcoin [74]. Due to its decentralized nature, a blockchain provides in this first use-case a means to transfer cryptocurrency without the control of a central body and thus, overcomes the cost of intermediaries (such as bank fees, taxes, etc.) during transactions. A blockchain is an immutable, distributed ledger that is not only used to store cryptocurrency transaction logs but can also be applied to numerous domains, including business, healthcare, passport-verification, and others [1, 32], for storing data, secured with SHA-256 hashing cryptography algorithms. Consensus algorithms such as **proof-of-work (PoW)** and **proof-of-stake (PoS)** play an essential role in preventing unauthorized modifications of highly secure transactions between untrusting peers through verification and validation.

Transaction verifications in PoW are computationally expensive and the probability of receiving rewards (viz. cryptocurrency) is based on a first-come-first-served principle. That is, the node that solves a necessary cryptographic puzzle first is declared as a winner, whereas in PoS the probability of receiving rewards is higher for nodes that have more cryptocurrency to stake.

A smart contract utilizes a blockchain to empower organizations by automating business processes [64]. Thus, a smart contract is a computer-program that enforces agreement rules automatically without relying on intermediaries. In his work from 1996, Nick Szabo [97] explains the concept of smart contract as a vending machine, in which parties participate in an exchange with the vendor via a coin for which the security is assured by a lock-box, or security mechanism. Hence, according to Szabo, smart contracts can also enable the exchange of valuable property controlled by digital means. Although the concept of smart contracts was proposed well before the 2000s, it is the inherent decentralization of blockchain technology that renders the implementation of smart contracts possible, as contract owners participating in an exchange via blockchain technology can do so without the need of trust.

Smart contracts are written in high-level languages (e.g., Solidity), and then intermediate languages such as Simplicity [102] and Scilla [93] are used for program analysis and verification. The latter offers strong security guarantees by utilizing type-soundness. Smart-contract code written in these languages can be executed on virtual machines (e.g., **Ethereum Virtual Machine (EVM)**) that require low-level instructions. Just as in the case of Ethereum, several blockchains also implement their own virtual machines, such as Rootstock,[4] **Telegram Open Network (TON)**,[5] and Bitcoin.[6] For instance, Rootstock introduces the Rootstock Virtual Machine to complement Bitcoin [41], whereas TON implements the **TON Virtual Machine (TON VM)** or **Telegram Virtual Machine (TVM)** for creating, managing, and debugging smart contracts [37].

2.2 Legal Implications of Smart Contracts

Consensus between parties is a required precondition to establish legally enforceable contracts. As Governatori et al. investigate in their work [49], the conceptual connection between legal contracts and smart contracts is that smart contracts must fulfill the necessary conditions such as offer and acceptance, consideration, competence, and capacity, and so on, to be legal contracts. Still, legal recognition in the interpretation of smart contracts depends on the paradigm of SCL [49]. Savelyev argues [90] that smart contracts comply with the Roman contract law. He explains this by comparing contracts to the mechanism of a vending machine and proposes solutions for alignment to the power of government with blockchain decentralization. The proposed solutions are based on providing the state authorities as a superuser the right to modify the blockchain databases while also relying on traditional remedies and enforcement practices. Additionally, Goldenfein et al. explain [47] that the legality of smart contracts depends on linking computational transactions to natural contracts, because the latter do not form an agreement themselves, while De Filippi et al. [29] define a smart contract as "law is code" and suggest shifting from the notion of "code is law." It is important to note that the semantics of contract law is lost while translating into smart-contract code [13], and thereby the status of legal recognition in smart contracts is not clear.

2.3 Recent Surveys

We list secondary research (see Table 1) that discuss the state-of-the-art in SCL and identify critical issues/challenges hindering the adoption of smart contracts. By investigating said research that

[4]Rootstock (RSK) home page.
[5]TON GitHub page.
[6]Bitcoin home page.

Table 1. Recently Published Studies and Surveys, Related to SCLs

Study Type	Author	Study Title	Year
SMS	M. Alharby et al.	A systematic mapping study on current research topics in smart contracts [4]	2017
	M. Alharby et al.	Blockchain-based smart contracts: A systematic mapping study [3]	2017
	D. Macrinici et al.	Smart contract applications within blockchain technology: A systematic mapping study [66]	2018
	F. Tariq, and P. R. Colomo	Use of blockchain smart contracts in software engineering: A systematic mapping [98]	2019
SLR	F. Casino et al.	A systematic literature review of blockchain-based applications: current status, classification and open issues [22]	2019
Surveys	P. L. Seijas	Scripting smart contracts for distributed ledger technology [92]	2016
	A. Nicola et al.	A survey of attacks on Ethereum smart contracts SoK [8]	2017
	G. Governatori et al.	On legal contracts, imperative and declarative smart contracts, and blockchain systems [49]	2018
	S. Wang et al.	An overview of smart contract: Architecture, applications, and future trends [103]	2018
	A. Miller et al.	Smart contracts and opportunities for formal methods [71]	2018
	V. Dwivedi et al.	Formal verification of smart-contract languages: A survey [38]	2019
	D. Harz and W. Knottenbelt	Towards safer smart contracts: A survey of languages and verification methods [52]	2018

is published as **systematic mapping studies (SMS)**s, SLRs and surveys, we are able to narrow down the research gaps and are therefore able to improve the quality of our research questions. Furthermore, through this analysis, we are also able to identify keywords and other related words used in the context of smart contracts and SCLs that are critical for the first phase of this SLR.

2.3.1 Systematic Mapping Studies. The focus of Alharby et al.'s SMS [4] is to identify existing gaps between blockchain technology and smart contracts, especially from different aspects of contract source codes and applications. The authors identify the research gaps of scalability and performance in smart contracts by investigating 24 studies and then categorize the open issues of codifying, privacy, security, and performance.

In their work, Macrini et al. [66] focus on smart-contract challenges and identify 64 papers to answer research questions that range from finding the research direction to questions related to approaches and identify problems and their solutions. Furthermore, they discuss the programmability of smart contracts without considering SCLs. In another SMS [98], the authors identify problems and solutions for usability and security of smart contracts and blockchains. The research questions are based on the general question of how software engineering can be applied in smart contracts. Eight primary studies are identified from scientific databases, and, based on these, the authors find that there is a lack of professional skills required for the development of smart-contracts and blockchains in software engineering.

2.3.2 *Systematic Literature Reviews.*

Research questions of SLRs are more concrete than those of SMSs. Although several SLRs (e.g., References [51, 53, 56, 100]) have been published, we find that none of these SLRs focus on smart contracts; rather, their research questions pertain to blockchain-based applications. While Casino et al.'s SLR [22] describes open issues of blockchain and smart-contract applications, the SLR does not focus on SCLs and therefore, the research questions are not aligned with smart contracts.

2.3.3 *Surveys.*

Table 1 presents existing surveys on smart contracts and based on analysis of said surveys, we find that none of the articles focus on smart contracts. Seijas et al. [92] provide an overview of scripting languages for several blockchain systems, such as NXT,[7] Bitcoin, and Ethereum, and critique languages with respect to security issues such as stack overflows in the context of distributed ledgers and cryptocurrencies. Atzei et al. [8] discuss a range of attacks that exploit the security vulnerabilities of the Ethereum smart contract. In Governatori et al.'s [49] work, the suitability of imperative- and declarative languages are examined by focusing on the lifecycle of legal contracts, and the authors discover that declarative SCLs are more suitable for developing legal smart-contract code. Still, the survey only focuses on a theoretical point of view. Wang et al. [103] discuss the recent advances and future trends in smart contracts but explicitly consider Ethereum and Hyperledger.[8] Miller et al. [71] present various security analysis tools to prevent attacks on smart-contracts and identify potential challenges. Harz et al. [52] provide an overview of existing SCLs focusing on security features. Finally, in our previous work [39], we analyze the suitability of existing SCLs for business collaboration and present the results through informed arguments.

As we mentioned earlier, looking at secondary research discussed above, it is clear that although there is a rising interest in investigating smart contracts and SCLs, unfortunately, current research is more focused toward security, scalability, and performance issues. This provides us with the opportunity to contribute to the current body of research by exploring smart contracts and SCLs from a legal standpoint.

3 REVIEW METHODOLOGY

Drawing from the above-mentioned gaps in the literature, we conduct this SLR and attempt to answer the research questions presented in Section 1. We choose this methodology, as SLRs are a transparent means of aggregating knowledge from available literature, based on an unbiased, auditable, and repeatable methodology. In our work, we follow the recommendations and guidelines provided by Kitchenham et al. [59] and therefore conduct this research in five stages,

- Study Identification
- Study Selection
- Quality Assessment
- Data Extraction
- Study Synthesis

[7]https://www.jelurida.com/nxt.
[8]Hyperledger home page.

Table 2. Search Terms Identified Based on Research Questions

Primary Search Keywords	blockchain, "smart-contract language," "smart contract languages," "smartcontract language"
Secondary Search Keywords	verification, "business process," legal, expressiveness, enforceable, rights, obligation, semantics, "decentralized autonomous organization," collaboration
Search Query	("smart contract" AND language*) AND (verification OR "business process" OR legal OR expressiveness OR enforceable OR rights OR obligation OR semantics OR blockchain OR "decentralized autonomous organization" OR collaboration)

Table 3. Study Sources and Identified Paper Count

Search Engine	Databases	Article Count
Scopus*	including ACM DL, IEEE Xplore, Elsevier, Springer	160
Web of Science*	including ACM DL, IEEE Xplore, Elsevier, Springer	50
Google Scholar*+	*including grey literature*	330
GitHub++		76
	Total	616
	Total (after duplicate removal)	406

* Journals, proceedings, and book chapters.
+ Technical reports, white papers, and working papers.
++ Only SCL documentation.

3.1 Study Identification

With the initial understanding of SCLs (from SMSs, SLRs, and surveys discussed in Section 2.3) and based on our research questions, we first identify the keywords typically used when describing SCLs. We divide the study-identification phase into three steps, namely, creation of search string, selection of study sources, and the search process. The search keywords we identify are based on the research questions and are then used to generate the search string (see Table 2). The academic databases and websites that we use to identify relevant work are detailed in Table 3.

3.1.1 Search Keywords. Since the focus of our research questions are blockchain smart-contract languages, we use "blockchain" and "smart contract language" as primary search keywords. Based on our experience looking at previous literature, we also include other variants of these keywords in the primary search keywords.

The secondary search keywords are more specific to individual research questions, and thus they include keywords such as verification, legal, decentralized autonomous organization, and their synonyms. Next, we generate the search string by applying boolean 'AND' and 'OR' operators on the primary- and secondary search keywords and run the queries in the next step.

3.1.2 Study Sources. Since the primary goal of this SLR is to identify and analyze SCLs and related work published in both academic and non-academic venues, we decide to search through a wide variety of sources, including well-renowned scholarly databases such as Web of Science, Scopus, arXiv, and Google Scholar, and industry focused online publishing platforms including GitHub. Through the former, we identify the necessary white literature, whereas the latter is useful for identifying the grey literature published as white papers, technical-reports, and SCL documentation.

Table 4. List of inclusion criteria

Criteria ID	Inclusion Criteria
IC1	The article discusses legal challenges in smart-contract languages.
IC2	The article addresses formal verification of languages.
IC3	The article proposes new insights for business-contractual supportive languages.
IC4	The article describes methods, or tools that enhance semantics of smart-contract languages.
IC5	The article answers the suitability of business-collaborative languages.
IC6	The article provides a comparative study of smart-contract languages.

Table 5. List of exclusion criteria

Criteria ID	Exclusion Criteria
EC1	The article describes smart-contract applications.
EC2	The article presents a survey of smart contracts.
EC3	The article discusses mechanisms based on cryptocurrencies.
EC4	The article describes technical details *(including challenges)* of smart-contract infrastructures.
EC5	The article presents a review of security aspects for smart contracts.
EC6	The article discusses the use of SCLs in particular use cases.

3.1.3 Search Process. To identify relevant literature and prevent redundancies in the search process, we categorically ascertain related articles from different study sources, one at a time. We begin by running the search query on Web of Science and Scopus and then import the list of articles into the Mendeley reference management software. Using Mendeley, we remove duplicate and redundant articles to then run the search query again in Google Scholar. We discover that several previously identified articles are also available in Google Scholar. Thus, we remove said articles manually, and in total, we are able to identify 540 scholarly articles published between 2015 and 2019.

At this stage, we run our search string on GitHub and identify 76 repositories in which codes and documentations for various SCLs are available. Through this process, we are able to identify a total of 28 studies (see Table 3 for details).

3.2 Study Selection

To further remove less relevant articles identified in the previous steps, in this phase of the SLR, we use a two-phase screening process, namely, the inclusion- and exclusion phase.

To identify relevant primary studies, we use a two-pronged approach. Using a predefined set of inclusion- and exclusion criteria, we aim to reduce the number of identified articles. To achieve this, we first examine the titles and abstracts for the identified articles and then verify if each of the articles fulfills the inclusion- and exclusion criteria stated in Table 4 and Table 5, respectively. By doing so, we are able to identify 130 primary studies (see Table 1 in Appendix A.2) that explicitly discuss SCLs from a suitability-, verifiability- and legality perspective and do not discuss security vulnerabilities and smart-contract applications. Furthermore, to verify the completeness of the SLR, and to assure that no relevant literature is omitted, we examine the references and footnotes of all identified studies. With this step, we are also able to cross-reference the previously identified

Table 6. List of quality assessment criteria for selected studies (QC1) and supporting studies (QC2)

Criteria ID	Quality Criteria
QC1.1	Is the proposed language named in the article?
QC1.2	Is the discussed SCL associated to a specific blockchain?
QC1.3	Is the purpose of the proposed language presented in the article?
QC1.4	Does the article discuss the paradigm of the language?
QC1.5	Is the source code of the SCL freely available?
QC1.6	Is the type system of the SCL described in the article?
QC2.1	Does the article discuss the specification of an SCL?
QC2.2	Does the article discuss the semantics of a business-collaboration language?
QC2.3	Are approaches for smart-contract verification discussed in the article?
QC2.4	Does the article detail legal semantics of the proposed SCL?

Table 7. Number of articles selected from each selection phase

Search Stages	Identified Articles
Study identification phase	616
	406 *(after screening duplicates)*
Study selection phase (I and II)	130 primary studies
Quality assessment phase	45 selected studies & 28 supporting studies

grey literature (including GitHub pages) with related scientific articles and are therefore able to validate if the selected grey articles are truly related to our research questions.

3.3 Quality Assessment

In this phase of the study-selection process, all authors meticulously read through the 130 primary studies selected in the previous stage. We then independently score each of the studied articles, according to a predefined set of quality measures referred to as quality-assessment criteria (see Table 6). The quality-assessment phase is a critical part of the SLR methodology and assists in removing the primary studies that, at first glance, appear important for the research questions but are actually not. At the end of the quality-assessment phase, we identify 73 primary studies (see Table 7), which we refer to as selected studies and supporting studies in Section 3.4. The selected scientific articles and grey literature are indicated with IDs SS1–SS46 and GL1–GL27, respectively (see Tables 9 and 11 for details).

3.4 Data Extraction

In this phase of the SLR, we extract necessary data from the selected- and supporting studies as per the quality-assessment phase, using the extraction form, presented in Table 8. The extraction form is used to acquire two different sets of knowledge from the studies: (i) data related to publication quality and (ii) data required to answer the research questions (RQ1, RQ2, and RQ3). The extracted details of the selected- and supporting studies are enumerated in Tables 9, 10 and 11, while the data specific to RQs are detailed in Sections 4 and 5.

The publication-quality details extracted from this step are presented in Appendix A.2, together with the details of all identified primary studies. Also, Figure 1 (in Appendix A.1) illustrates the number of primary studies (both white- and grey literature) published between 2015 and 2019.

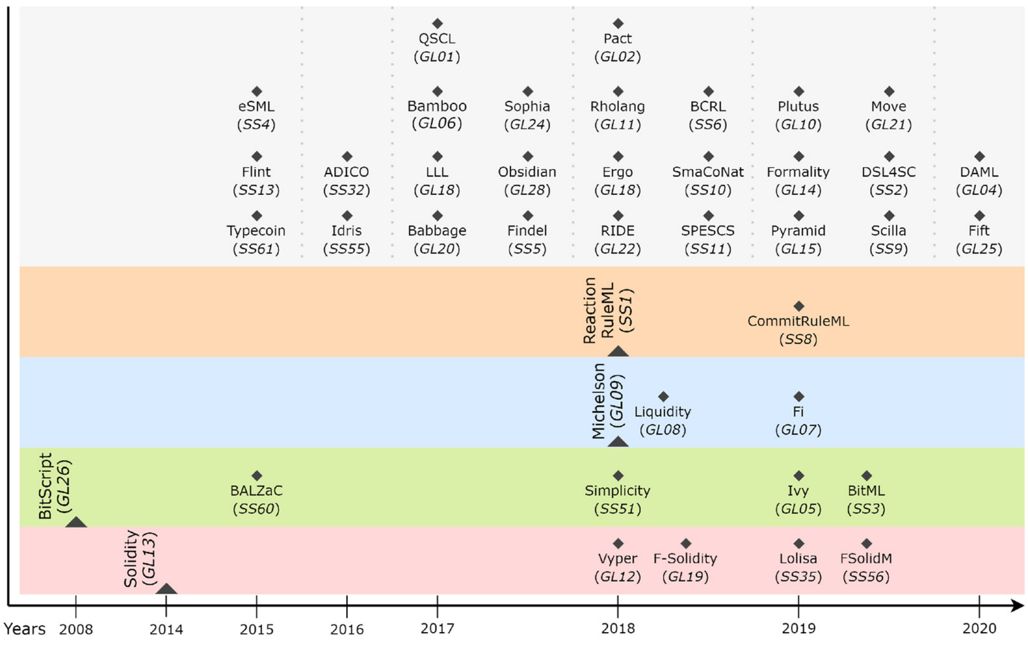


Fig. 1. SCL Implementations per Year.

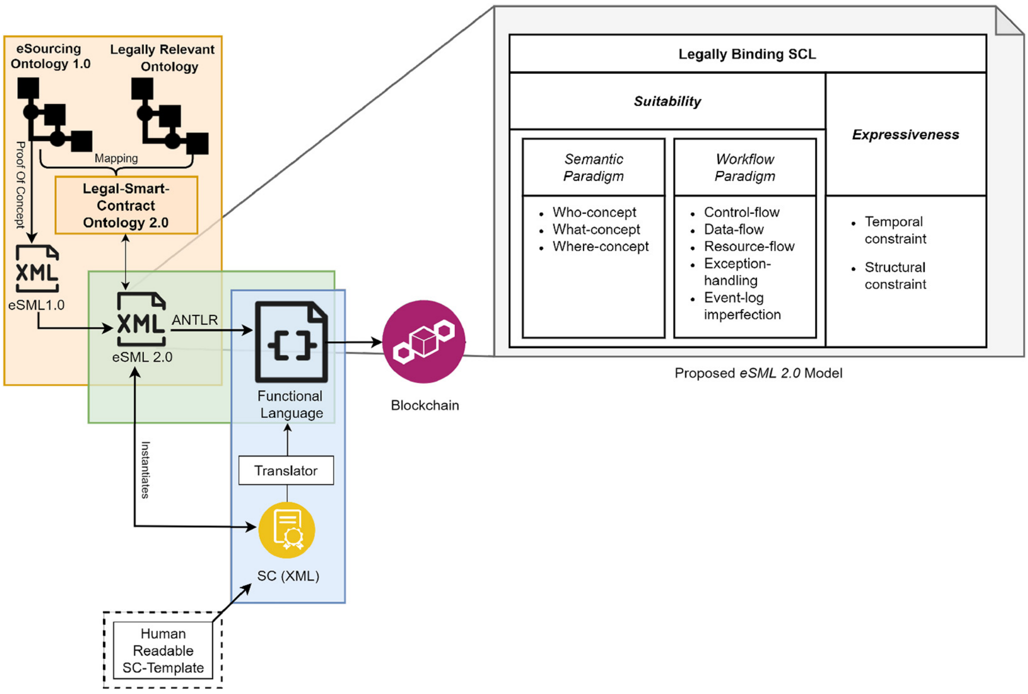


Fig. 2. Proposed Model for the Development of Legally Binding SCL.

Table 8. Data Extracted from Selected and Supporting Studies

Source Type	Extracted Data	Details
Publication	Article's Title, Author's Name, Publication Type, Year, Publisher, and Citation Count	Tables 9, 11 and Appendix A.2
Smart Contract Language	Name, Blockchain, Type System, Paradigm, Focus, and Purpose	RQ1
	Legal Semantics, Business Semantics, Business Objectives, and Expressiveness	RQ2
	Challenges	RQ3

4 SUMMARY OF SELECTED STUDIES

This section synthesizes the knowledge presented in the selected studies and describes the identified SCLs. We examine the said SCLs and summarize their attributes, namely, the *name* of the SCL, the *blockchain* platform it is designed for, and its *type-system*, *paradigm*, *focus*, and *purpose*. We choose these attributes as they are critical for understanding the differences between the state-of-the-art SCLs and for identifying critical properties that can enable the drafting of legally binding smart-contracts.

The attributes *focus* and *purpose* indicate the motivation behind the development of SCL, i.e., the particular domain/activity the SCL was designed for, and whether the proposed SCLs are detailed enough to be checked for semantic correctness, or formal verification, or both. The *paradigm* attribute of SCLs refers to its programming paradigm, or execution model (viz. process-flow and data-flow). Finally, the *type-system* of SCLs defines the rules that apply to the data types in the programming language. These attributes are specially important, as they provide critical insights into the suitability and expressiveness of SCLs. In particular, *focus* and *purpose* provide insights about semantic suitability, *focus* and *paradigm* provide insights about workflow suitability, and *purpose* and *type-system* provide insights into expressiveness.

- *RQ1*: What blockchain-based SCLs already exist in scientific and non-scientific literature?

As mentioned in Section 3.3, after applying the quality-assessment criterion to the primary studies, we are able to identify 45 selected studies (each proposing a novel SCL). Since the number of selected SCLs is too large to be described individually, we segregate them into five categories based on the foci of the SCL; namely, we categorize them as *domain-specific* SCLs, *formally verifiable* SCLs, *easy-to-use* SCLs, *legally enforceable* SCLs, and *business process* SCLs.

SCLs that are designed with an emphasis on specific domains are categorized as *domain-specific* SCLs. *Formal verifiable* SCLs include languages that are designed with priority to runtime safety of smart-contract code. *Easy-to-use* SCLs are simply languages that are human understandable, or lay-person friendly. *Legally enforceable* SCLs are the ones that are designed to (or aim to) create legally binding contracts. Last, *business-process* SCLs are the languages that are designed with an emphasis on business-process automation.

4.1 Domain-Specific SCLs

Solidity (GL13) is regarded as a domain-specific language designed to compose complex smart contracts for digital assets such as voting, crowdfunding, and so on [94]. Solidity is a statically typed language that supports multiple inheritance and complex user-defined data types. Contracts are represented as **finite state machines (FSM)s** in Solidity, preventing any transaction call to other

Table 9. List of Selected Studies

PS ID$^+$	SS ID$^+$	Study Title	Year	P.Type*
PS1	SS1	An introduction to commitment based smart contracts using ReactionRuleML	2018	WP
PS2	SS2	Automatic smart contract generation using controlled natural language and template	2019	JA
PS3	SS3	BitML: A calculus for bitcoin smart contracts	2018	CP
PS4	SS4	eContractual choreography-language properties toward cross-organizational business collaboration	2015	JA
PS5	SS5	Findel: Secure derivative contracts for ethereum	2017	JA
PS6	SS6	Empowering Business-Level Blockchain Users with a Rules Framework for Smart Contracts	2018	JA
PS7	SS7	Towards Automated Generation of Smart Contracts	2016	CP
PS8	SS8	Introducing commitRuleML for smart contracts	2019	WP
PS9	SS9	Safer Smart Contract Programming with Scilla	2019	JA
PS10	SS10	SmaCoNat: Smart Contracts in Natural Language	2018	CP
PS11	SS11	SPESC: A Specification Language for Smart Contracts	2018	CP
PS12	SS12	Towards Adding Variety to Simplicity	2018	CP
PS13	SS13	Writing safe smart contracts in Flint	2018	CP
PS14	SS14	Building a blockchain simulation using the Idris programming language	2019	CP.
PS15	SS15	Tool Demonstration: FSolidM for designing secure ethereum smart contracts	2018	CP
PS16	SS16	Marlowe: Financial contracts on blockchain	2018	CP
PS17	SS17	Peer-to-peer affine commitment using Bitcoin	2015	CP
PS41	SS18	Bitcoin Covenants.	2016	CP
PS18	GL1	Smart-Contract Value-Transfer Protocols on a Distributed Mobile Application Platform	2017	TR
PS19	GL2	The Pact smart contract language	2017	TR
PS20	GL3	Bitcoin Abstract Language, analyZer and Compiler	2018	TR
PS21	GL4	DAML SDK Documentation	2019	TR
PS22	GL5	Ivy : A Declarative Predicate Language for Smart Contracts Introduction : Two Blockchain Models	GitHub	
PS23	GL6	Bamboo: a language for morphing smart contracts	2017	GitHub
PS24	GL7	fi - Smart Contract language for Tezos	2019	GitHub
PS25	GL8	Welcome to Liquidity's documentation! — Liquidity 1.0	2018	GitHub
PS26	GL9	Michelson : the language of Smart Contracts in I - Semantics	2018	GitHub
PS27	GL10	Formal Specification of the Plutus Core Language.	2019	GitHub
PS28	GL11	Contracts, Composition, and Scaling: The Rholang specification 0.2.	2018	GitHub
PS29	GL12	Vyper Documentation.	2018	GitHub
PS30	GL13	Solidity Documentation Ethereum	2016	GitHub
PS31	GL14	Formality Documentation Release 0.3.157	2019	GitHub
PS129	GL15	Pyramid Scheme Introduction	2017	GitHub
PS130	GL16	LLL Compiler Documentation Documentation Release 0.1.	2017	GitHub

(Continued)

Table 9. Continued

PS ID+	SS ID+	Study Title	Year	P.Type*
PS32	GL17	Functional-solidity-language	2017	GitHub
PS33	GL18	Babbage — a mechanical smart contract language	2017	GitHub
PS34	GL19	Mutan smart contract Language	2015	GitHub
PS35	GL20	ErgoScript, a Cryptocurrency Scripting Language Supporting Non interactive Zero-Knowledge Proofs	2019	GitHub
PS36	GL21	Move : A Language With Programmable Resources.	2019	GitHub
PS37	GL22	Ride: a Smart Contract Language for Waves	2019	GitHub
PS38	GL23	IELE: An Intermediate-Level Blockchain Language Designed and Implemented Using FormalSemantics.	2018	GitHub
PS39	GL24	Sophia Introduction	2017	GitHub
PS40	GL25	Fift : A Brief Introduction	2017	GitHub
PS42	GL26	Lolisa: Formal Syntax and Semantics for a Subset of the Solidity Programming Language	2018	ePrint
PS43	GL27	Obsidian: Typestate and Assets for Safer Blockchain Programming	2019	ePrint

+ *PS ID*: Primary Study ID. *SS ID*: Supporting Study ID.

* *P.Type*: Publication Type. *WP*: Workshop Proceedings. *CP*: Conference Proceedings. *JA:* Journal Article. *TR:* Technical Report.

contracts within a state transition. A transaction sent to a contract that changes the state is either successful, or raises an exception. Solidity has several bugs and vulnerabilities, such as *re-entrancy* and *delegatecall*; several tools or frameworks, such as Oyente, Mythril, Securify, and others, have been developed to verify and analyze Solidity code [72]. Ethereum introduces the Vyper language (GL12) [72] that is developed to overcome the Solidity bugs by focusing on security, audibility, and simplicity. Vyper aims to make it harder for the developer to write malicious code intentionally. Furthermore, Vyper prevents unintended security vulnerabilities in the code by using the inbuilt libraries of integer (*overflow/underflow*). Solidity does not use any checks on return values of transactions due to exceptions occurring in the caller contract. In contrast, Vyper provides two ways to handle such exceptions: by implementing *send()* and *raw-call()* functions; using these the entire transaction is reverted, in case of failures. Moreover, Vyper supports functions for preventing the re-entrance vulnerability, when a contract calls to an external contract. A contract is particularly vulnerable to a re-entrance attack if necessary state changes do not occur before calling an external contract. Vyper implements the *nonreentrant decorator*[9] that places a lock on the current function and all functions with the same key value. The SCL Idris (SS14) has been developed to secure the smart-contract code by implementing *dependent types* [50]. Idris allows types that depend on values, i.e., types are first-class language constructs and can be manipulated comparable to any other values.

Flint (SS13) is a contractual-specific and statically typed SCL that compiles into EVM bytecode through the intermediate language [79] YUL.[10] The Solidity team designs YUL to support several EVM backends such as EVM 1.0 and EVM 1.5 and multiple languages as a front-end. Flint aims to write inherently safer and predictable smart-contract code, rather than analyzing the code after being written, as in Solidity. To prevent unauthorized calls to contract-sensitive functions, Flint uses a *caller capability block* in which the right to call Ethereum user accounts is declared. Flint pro-

[9] Vyper | Documentation on Structure of a Contract.

[10] Yul | GitHub Page.

Table 10. Details of SLCs Presented in Selected Studies

SCL	Study ID	Blockchain	Type-System	Paradigm	Purpose*	Focus+
Reaction-Rule ML	SS1 [30]	—	Static	Declarative	SPEC	Legal Contracts
DSL4SC	SS2 [99]	Hyperledger	Dynamic	Declarative	SPEC	Natural Language
BitML	SS3 [11]	Bitcoin	Dynamic	Declarative	IMPL	Security
eSML	SS4 [77]	—	Dynamic	Declarative	SPEC	Business Process
Findel	SS5 [15]	*Independent*	Dynamic	Declarative	SPEC	Financial Contracts
BCRL	SS6 [7]	Hyperledger	Dynamic	Declarative	IMPL	Business Process
ADICO	SS7 [44]	Ethereum	Dynamic	Declarative	SPEC	Legal-Contracts
Commit-Rule ML	SS8 [31]	—	Static	Declarative	SPEC	Legal-Contracts
Scilla	SS9 [94]	Zilliqa	Static	Functional	IMPL	Security
SmaCoNat	SS10 [84]	—	Dynamic	Imperative	SPEC	Natural Language
SPESCS	SS11 [54]	—	Dynamic	Declarative	SPEC	Legal-Contracts
Simplicity	SS12 [102]	Bitcoin	Dynamic	Functional	IMPL	Security
Flint	SS13 [91]	Ethereum	Static	Imperative	IMPL	Security
Idris	SS14 [79]	Ethereum	Dependent	Declarative	IMPL	Security
FSolidM	SS15 [69]	*Independent*	Dynamic	Declarative	SPEC	Security
Marlowe	SS16 [61]	Cardano	Dynamic	Declarative	SPEC	Financial Contracts
Typecoin	SS17 [27]	Bitcoin	Type-Safety	Symbolic	IMPL	Crypto.
QSCL	GL1 [28]	Qtum	Static	Imperative	IMPL	Business Process
Pact	GL2 [83]	Kadena	Dynamic	Declarative	IMPL	Security
BALZaC	GL3 [9]	Bitcoin	Dynamic	Imperative	SPEC	Verification
DAML	GL4 [34]	Hyperledger	Dynamic	Declarative	IMPL	Business Process
Ivy	GL5 [86]	Bitcoin	Static	Declarative	IMPL	Domain Specific
Bamboo	GL6 [110]	Ethereum	Type-Safety	Imperative	IMPL	Formal Verf.
Fi	GL7 [5]	Tezos	Type-Safety	Imperative	IMPL	Verification
Liquidity	GL8 [78]	Tezos	Dynamic	Functional	IMPL	Formal Verf.
Michelson	GL9 [101]	Tezos	Monomorphic	Low-Level	IMPL	Domain Specific
Plutus	GL10 [21]	Cardano	Dynamic	Declarative	IMPL	Financial Contracts
Rholang	GL11 [70]	Rchain	Dynamic	Declarative	IMPL	Domain Specific

(Continued)

Table 10. Continued

SCL	Study ID	Blockchain	Type-System	Paradigm	Purpose[*]	Focus[+]
Vyper	GL12 [20]	Ethereum	Dynamic	Imperative	IMPL	Security
Solidity	GL13 [43]	Ethereum	Static	Imperative	IMPL	Domain Specific
Formality	GL14 [68]	Ethereum	Static	Declarative	IMPL	Efficiency
Pyramid	GL15 [19]	Ethereum	Strongly Typed	Imperative	IMPL	Safety
LLL	GL16 [40]	Ethereum	Dynamic	Declarative	IMPL	User Frnd.
F-Sol	GL17 [85]	Ethereum	Static	Functional	IMPL	Verification
Babbage	GL18 [24]	Ethereum	Type-Safety	Symbolic	SPEC	Human Undr.
Mutan	GL19 [106]	Ethereum	Dynamic	Imperative	IMPL	Formal Verf.
ErgoScript	GL20 [33]	*Independent*	Type-Safety	Declarative	SPEC	Legal-Contracts
Move	GL21 [16]	Libra	Static	Imperative	IMPL	Verification
RIDE	GL22 [12]	Waves	Static	Declarative	IMPL	User Frnd.
IELE	GL23 [58]	IELE	Static	Imperative	IMPL	Verification
Sophia	GL24 [107]	Aeternity	Strongly Typed	Imperative	IMPL	Domain Specific
Fift	GL25 [37]	TON	Dynamic	Imperative	IMPL	Domain Specific
Script	SS18 [73]	Bitcoin	Static	Imperative	IMPL	Crypto.
Lolisa	GL26 [109]	Ethereum	Static	Imperative	SPEC	Formal Verf.
Obsidian	GL27 [26]	Hyperledger	Static	Imperative	IMPL	Security

[*] *SPEC*: Specification. *IMPL*: Implementation.
[+] *Verf.* [Verifiable] | *Frnd.* [Friendliness] | *Undr.* [Understandable] | *Crypto.* [Cryptocurrency].

vides safer atomic transactions and ensures that the contract state is consistent. Formality (GL14) is more time-efficient than the Solidity in contract execution, because its core is built as an affine lambda calculus that allows it to be garbage-collection free [68]. Formaility is statically typed and resembles a Python-like programming language featuring formal proofs. Pyramid Scheme (GL15) is a functional and imperative SCL, promoting separation of state-changing and static functions. Also, Pyramid scheme targets the EVM [19]. The functions are designed to be atomic and executed completely to ensure consistency in smart-contract state changes. In addition, pure functions are used in Pyramid Scheme to indicate that there is no effect on the global and local states. Findel (SS5) is a declarative- and domain-specific SCL that is developed for handling financial agreements securely and designed for targeting the EVM. Findel is a formal language that restricts the formalization of unambiguous contractual clauses [15] by separating the description of the contract from its execution.

Michelson (GL9) is a low-level stack-based language developed by Tezos [101]. Michelson's instructions are executed with unrestricted stack-length that ensures the secure execution of the Michelson code. Michelson is different from Solidity relating to smart contracts that aim to write a piece of business logic. Ethereum contracts are written to implement concepts such as multi-sig wallets, vesting, distribution rules, and so on, and Michelson is not only for writing arbitrary

Table 11. List of Supporting Studies

PS ID+	SS ID+	Study Title	Year	P.Type*
PS45	SS19	A Method of Logic-Based Smart Contracts for Blockchain System.	2018	CP
PS46	SS20	A Solution for the Problems of Translation and Transparency in Smart Contracts.	2017	TR
PS47	SS21	Auto-Generation of Smart Contracts from Domain-Specifc Ontologies and Semantic Rules.	2018	CP
PS49	SS22	Conflict-Resolution Lifecycles for Governed Decentralized Autonomous Organization Collaboration.	2015	CP
PS50	SS23	Contract law 2.0: ‘Smart’ contracts as the beginning of the end of classic contract law	2017	JA
PS51	SS24	Creation of Smart-Contracting Collaborations for Decentralized Autonomous Organizations	2015	CP
PS52	SS25	Debugging Smart Contract’s Business Logic Using Symbolic Model Checking	2018	ePrint
PS53	SS26	Designing a Smart-Contract Application Layer for Transacting Decentralized Autonomous Organizations	2017	CP
PS54	SS27	Developing secure bitcoin contracts with BitML	2019	JA
PS55	SS28	Evaluation of Logic-Based Smart Contracts for Blockchain Systems	2016	CP
PS56	SS29	FEther: An Extensible Defnitional Interpreter for Smart-Contract Verification in Coq	2019	JA
PS57	SS30	Formal modeling and verification of smart contracts	2018	CP
PS58	SS31	Formal Requirement Enforcement on Smart Contracts Based on Linear Dynamic Logic	CP	
PS59	SS32	Formal Specification Technique in Smart Contract Verification	2019	CP
PS60	SS33	Formal Verification of Blockchain Smart Contract Based on Colored Petri Net Models	2019	CP
PS61	SS34	Formal verification of smart contracts: Short paper	2016	CP
PS63	SS35	Legally speaking Smart contracts, archival bonds, and linked data in the blockchain	2017	CP
PS64	SS36	Making Smart Contracts Smarter	2016	CP
PS65	SS37	On legal contracts, imperative and declarative smart contracts, and blockchain systems	2018	JA
PS66	SS38	Smart Contract Programming Languages on Blockchains: An Empirical Evaluation of User-friendly and Security	2018	Book

(Continued)

Table 11. Continued

PS ID+	SS ID+	Study Title	Year	P.Type*
PS67	SS39	Smart Contracts and Opportunities for Formal Methods.	2018	CP
PS69	SS40	SmartCheck: Static analysis of ethereum smart contracts.	2018	CP
PS120	SS41	SMT-based verification of Solidity smart contracts	2018	JA
PS72	SS42	Towards a Shared Ledger Business Collaboration Language Based on Data-Aware Processes.	2016	CP
PS73	SS43	Towards Safer Smart Contracts: A Survey of Languages and Verification Methods.	2018	ePrint
PS74	SS44	A Formal Model of Bitcoin Transactions.	2018	CP
PS122	SS45	A Survey of Attacks on Ethereum Smart Contracts (SoK)	2017	CP
PS44	SS46	Safer smart contracts through type-driven development Using dependent and polymorphic types.	2016	MT

+ *PS ID*: Primary Study ID. *SS ID*: Supporting Study ID.
* *P.Type*: Publication Type. *WP*: Workshop Proceedings. *CP*: Conference Proceedings. *JA:* Journal Article. *TR:* Technical Report.

programs rather targeted to said applications. Plutus core (GL10) is invented for the use of transaction validation on a blockchain. Plutus Core is implied to be a compilation target expressed in the design of the language: even as writing large Plutus Core programs by hand is challenging, the language is relatively straightforward to formalize with a proof assistant [21]. Plutus-core is used for on-chain transaction validation. The validation process is out of the scope of this article as we refer the reader for further details to Reference [55]. Rchain implements Rholang (GL11) [70], a contractual- and concurrent-oriented programming language that supports contractual behavior by focusing on the concurrent data and process flow. Rholang is process-oriented, i.e., all communication is done through message passing. Rholang is equipped with a behavioral type-system and enables participants seeking to engage in a contractual agreement with a way of exploring contractual obligations and guarantees in an automated manner.

Marlowe (SS16) [61] is a domain-specific language targeting the execution of financial contracts. Marlowe is implemented as an algebraic type in Haskell programming on **Unspent transaction output (UTxO)**, or account-based blockchains. Pact [83] (GL2) is a declarative programming language with a lisp syntax and Haskell-comparable types that support programmers to implement less buggy code. Pact aims to enforce business rules guarding the update of system records stored on the Kadena blockchain. Sophia (GL24) [107] is blockchain-specific and strongly typed to support ML-comparable programming that has a restricted mutable state. Sophia aims to support blockchain-specific primitives, constructions, types, and specially designed for the private *Aeternity* blockchain. Fift (GL25) [37] is a stack-based programming language specially designed to create, manage, and debug smart contracts for the TON blockchain. Fift is a dynamic type language that keeps the value of different types other than an integer in the stack, is used for interactive experimentation, debugging, and writing simple scripts. Ivy [86] (GL5) is a predicate language designed for writing smart contracts for Chain VM and is intended for educational purposes. Bitcoin uses a Script (SS18) language [73], is a list of instructions recorded with each transaction that describes how they can be accessed by the next individual who wants to spend the Bitcoins being transferred. Typecoin (SS17) [27] language is a rational process of commitment designed to carry rational propositions on top of Bitcoin. The underlying concept of Typecoin is that a transaction is carrying logical proposals instead of coins. There is a possibility to translate each Bitcoin transaction into a Typecoin transaction where inputs and outputs become propositions, and the logic would allow inputs to be split, or merged.

4.2 Formally Verifiable SCLs

A smart contract can explicitly comprise rights and obligations semantics, similarly to conventional contracts. Many law loopholes exist in smart contracts due to the lack of common understanding among programmers and legal experts. These loopholes are different from common bugs that are easily discovered. Several static- and dynamic instruments have been developed and have not yet been proven to secure smart contracts such as Mythril, Oyente [14], and so on. Thus, the formal verifiability of a language is essential to ensure the correctness and runtime safety of the smart-contract code. The first mechanized and validated formal syntax and semantics developed for Solidity is Lolisa (GL26) [109], which supports not only solidity syntax such as mapping, modifiers, but also includes conventional programming characteristics such as multiple return types, structures, and so on. Besides that, Lolisa adopts a more robust static-type system than Solidity for enhancing type safety. Bamboo (GL6) [110] is a formally verified SCL to render transactions explicit that overcome the reentrancy behavior in the Ethereum contract. The programming language of the Bamboo enables reasoning as to state machines on smart contracts. In each state, developers define which functions can be called, and the language provides constructs to specify state changes explicitly.

Further, Ethereum implements Mutan (GL19) to support the dynamic feature of higher-level languages such as C, or C++ [106]. Move SCL [16] aims to encode the owners of digital assets and corresponding their business logic. Thereby, Move (GL21) provides governance rules for Libra blockchain[11] in a flexible, safe, and verifiable manner. The main aspect of Move is the ability to describe custom resource types with a semantics-inspired linear logic [46]: A resource can never be replicated, or tacitly discarded, only relocated between program storage locations. Furthermore, Move provides the flexibility of code by adding *transaction scripts*, which are single functions implemented once to invoke multiple modules published in blockchain procedures, allowing customizable transactions. Move fulfills vital security features such as memory safety, type safety, and resource safety by implementing *bytecode verifier* that checks the Move bytecode on-chain. FsolidM (SS15) [69] is a framework for visual programming and is used to define contracts as a FSMs. FsolidM implements a code generator for specifying FSMs, especially for creating Ethereum contracts. In addition, FsolidM offers a set of plugins used to enhance the security and functionality in FSMs. Plugins are designed to address the common design patterns of security vulnerabilities identified in prior work [8, 65]. Obsidian (GL27) is a state-oriented and static-type SCL that enables the developer to explicitly declare and transition states. Furthermore, Obsidian is based on core calculus that uses a *typestate* to detect incorrect state manipulation and *lineartypes* is used to ensure the resources are managed correctly by the program. Moreover, Obsidian supports Hyperledger Fabric, a permisisoned blockchain platform.

Fi (GL7) is a statically typed language, designed to be syntactically similar to Javascript, Solidity and directly compiles to Michelson code of the Tezos blockchain. The latter provides a familiar coding environment that is closer to an object-oriented programming language. Scilla (SS9) is an

[11]Libra home page.

intermediate language developed by *Zilliqa blockchain*[12] that is used as a translation target of high-level languages for performing program analysis and -verification. Scilla aims to achieve expressivity and tractability that allows contract behavior to be formally reasoned about. The design principle of Scilla is based on the separation between computation and communication, meaning computing a value of the function is implemented as a stand-alone. The design principle of Scilla is based on the separation between computation and communication, meaning computing a value of the function is implemented as a stand-alone. That means Scilla does change a balance without involving any other parties if involvement is required (e.g., transferring control to other parties), a transition would end by employing sending- and receiving messages. The existing languages have a separate specification and implementation, and if the performance is distinct, it is impossible to execute test cases against the specification. IELE (GL23) aims to bridge the gap between specification and implementation using K-framework.

BitML (SS3) is a Bitcoin modeling language where process calculi is used to stipulate the contract for regulating transfers of coins, establishing a smart contracts' correctness required for proving the computational security in the cryptographic protocol, which is an additional burden for the programmers. Therefore, BitML defines a symbolic- and computational model for reasoning about Bitcoin security. The semantic of BitML are defined in the symbolic model, and participants act accordingly. To reason about the participant's behavior, a computational model is used to introduce additional restriction for attackers. Simplicity (SS12) [102] is an SCL with formal semantics implemented in Haskell's functional programming language. Its type system supports multiple inheritance that enables the developer to express non-trivial contracts. The primary design goal of Simplicity is to offer the computation of runtime resource estimations statically. Liquidity (GL8) is a fully typed functional, high-level language implemented on an OCaml syntax [78]. Tezos blockchain invented the latter to replace the Michelson language, which is harder to read and write due to a lack of stack-based instructions. Liquidity uses a compiler to come up with Michelson and a decompiler for translating Michelson to Liquidity. Besides, Liquidity offers full coverage of Michelson language with additional local variables instead of stack manipulations. The functional Solidity (GL17) [85] language is developed to write Ethereum smart contracts with additional formal methods.

4.3 Easy-to-use SCLs

Lisp-Like-Language (LLL) is an intermediate-, simple- and minimalistic Ethereum SCL that translates the Solidity code (high-level) into bytecode (low-level) and makes it easy to use language by reducing the complexity of EVM stack-management. Besides, LLL (GL16) offers a different perspective than Solidity that does not hide the resource perspectives and allows limited resources to be used effectively. Moreover, LLL facilitates the creation of clean EVM code while directly removing the worst of the pain of coding, namely stack-and jump management for the EVM. Babbage (GL18) is a visual programming language developed [24] to understand complex smart-contract code without programming knowledge. Babbage aims to implement dataflow transparency by building a grasping an analogy of a vending machine. The language Ride (GL22) aims to overcome the shortcomings of existing languages by offering a straightforward, statically-typed functional language for distributed-application (DApp) development that pre-calculates the amount of gas required in a smart-contract execution [12]. DSL4SC (SS2) is developed to express structural- and temporal properties and constraints on states' sequences in a state machine. DSL4SC supports the Hyperledger blockchain and is used for specification purposes. SmaCoNat (SS10) [84] is a human-understandable and easy-to-use SCL in which a smart contract is stipulated in natural-language

[12]Zilliqa | Home Page.

syntax. The prepositions of SmaCoNat is used in natural language to define the properties of data structures. SmaCoNat is not a full-fledged featured language but focuses on smart-contract development in a natural language with small types and operations.

4.4 Legally Enforceable SCLs

We identify the SCL that can transform legal semantic rules into smart-contract code. The choreography languages such as ADICO (SS7) [44], ErgoScript (GL20) [33], CommitRuleML (SS8) [31], ReactionRuleML (SS1) [30], and SPESC (SS49) [54] are the foundation to design legal smart contracts. The ADICO framework proposes a modeling approach that allows the semi-automated translation of human-readable contracts to codify the laws into smart-contract. The ADICO programs are generated from several rule-based components, such as attributes, denotic, aim, conditions, and Or-else. The attributes component represents an actor's properties, and the denotic represents contract clauses such as rights, obligations, prohibitions. The aim defines the outcome of the ADICO program. On the contrary, the condition statement describes under which context this statement follows. Next, the Or-else components describe the prominences associated with a nonconformance. CommitRuleML regards a contract as commitment-based smart contracts that is an extension of the KR ReactionRuleML. The communication among parties is seen as a **multi-agent system (MAS)**, and the commitment between MAS an essential foundation for the interactions of the organizing parties. CommitRuleML defines the contract utilizing event calculus, and statements are defined, such as the events (On), condition (if), and actions (do) in executable language. ErgoScript is a scripting language that is specially implemented to support financial contracts and on-interactive zero-knowledge proofs. The latter enables the developer to encode the conditions under which currency is spent, e.g., who can pay and to whom, under what condition, and so on. ErgoScript is more potent than Bitcoin Script, because it does not contain a recursive construct that is an obstacle in estimating running time. SPESC is a specification language for specifying smart contracts similar to real-world arrangements using natural language in which the rights and obligations and the transaction rules are clearly defined. SPESC contains the specification, such as descriptions of parties, set of terms, and describing the transaction record when specific terms hold. Besides, SPESC (SS11) is intended to support the collaborative development of smart blockchain contracts. Still, these foundations are not mapped to high-level programming languages, e.g., Solidity, to implement legal smart contracts' semantics into code.

4.5 Business Process SCLs

Smart contracts enable the fulfillment of business processes on a blockchain. We identify the SCL such as DAML (GL4) [34], **business-level rules language (BCRL)** (SS6) [7], QSCL (GL1) [28], and **eSourcing Markup Language (eSML)** (SS4) [77] that incorporate the semantics of business processes while generating smart contracts. DAML is a functional programming language designed for permissioned blockchain, and its focus is to deal with the business processes rather than blockchain technology and encryption. The latter manages the rights and obligations by ensuring that affected parties can only view contract details through the contract. Besides, DAML restricts the instruction set, which is used to prevent unwanted behavior. Qtum blockchain develops QSCL that incorporates the semantic-web domain concept and properties. QSCL aims to promote the greater use of the value-transfer protocol management that organizes cross-organizational information logistics and value transfers in line with the value proposition. The specification of QSCL includes a structure for uniquely defining the contracting parties with the definition of resources and data. Further, the eSML is developed to support business collaboration by incorporating the contractual properties, such as party identification, business- and legal context, and exchange values. Thus, eSML is a choreography language for cross-organizational collaboration. A fully functional framework

BCRL is developed to implement business core logic in Controlled English language that helps in establishing a shared understanding among a domain expert and smart-contract developer. Besides, a BCRL smart contract is executed both as a smart contract on Hyperfabric ledger and in an off-chain rules engine, allowing a more seamless experience for managing general solutions for business collaboration.

As discussed above, we have found 17 SCLs from scientific literature and 28 SCLs from non-scientific literature. Of these, 28 SCLs are implemented, and the rest are proposed in academic articles. Figure 1 presents the year in which the SCLs are implemented/proposed with their associated study IDs.

5 SUITABILITY- AND EXPRESSIVENESS PROPERTIES

Using the *supporting studies* identified in the second phase of the study quality assessment (see Section 3.3), we next identify the indecomposable properties of SCLs that are critical in drafting legally-binding contracts. We do so by first conducting a thematic analysis of the 28 supporting studies (see Table 11) (using the NVIVO[13] qualitative data-analysis software) and then categorize said properties into three headings, "semantic suitability," "Workflow Suitability," and "expressiveness."

In their work, Mario et al. [18] provide a unifying theory that assists in verifying contracts' compliance in services choreography. Although the concept is primarily aimed toward the **service-oriented computing (SOC)** research community, it has significant implications in DAO research as well [76]. Just as services designed in SOC are "self-describing computational elements that support composition of distributed applications" [80], smart contracts in DAO too aim to be ontologically rich, formally verifiable pieces of code designed to support collaborations between decentralized entities. Drawing parallels between the two, it is not unfathomable to view the ontological concepts and properties of smart contracts [48, 82] as choreography-conformance requirements in service design [18]. In other words, said choreography-conformance requirements could be viewed as semantic and workflow properties (viz. *suitability*).

We define "semantic suitability" properties as fundamental components from the perspective of the eContract paradigm [77] and thus, includes properties that provide insights into the context, i.e., who is participating the in transaction, what are they exchanging, and under what provisions [49] of a smart contract. While "workflow suitability" encompass properties [89] that provide insights into the processes, i.e., how the transaction are being carried out, or workflow patterns from the perspective of contractual collaboration paradigm [77]. In our current work, we build on Norta et al.'s view of workflow patterns, and introduce Business Process Modeling patterns proposed by Russell et al. [89] as critical properties that can help understand how the state of the contract would change after any given set of interactions.

- *RQ2*: What properties of business-oriented SCLs constitute to suitability and expressiveness?

This RQ aims to identify and categorize the suitability- and expressiveness properties of SCLs based on the qualitative analysis of the supporting studies (see Table 11). As discussed earlier in Section 4, during our analysis of SCLs, we discover that SCLs are typically designed based on a broad spectrum of foci that influence the SCLs' formal verifiability and semantic correctness. Given this, it is difficult to identify all properties that render them legally binding from individual SCL supporting studies. Hence, we first accumulate all relevant properties from the supporting studies and then comprehensively examine said properties. Finally, we segregate them based on

[13]NVIVO | Home Page.

Table 12. Contractual Aspects Required for Legally Enforceable SCLs and Associated Supporting Studies

Aspects	Paradigm	Properties	Associated Studies
Suitability	Semantic	The Who-concept	SS23, SS28, SS36
		The Where-concept	SS35, SS42, SS46
		The What-concept	SS20, SS31
	Workflow	Control-flow	SS39, SS41
		Data-flow	SS34, SS45
		Resource-flow	SS29, SS30, SS44
		Exception-handling	SS32, SS43
		Event-Log Imperfection	SS33, SS34, SS37
Expressiveness		Temporal constraints	SS19, SS21, SS25, SS38
(*viz. formal-verification*)		Structural constraints	SS22, SS24, SS26, SS27

the previously defined categories, namely namely "semantic suitabilty," "workflow suitability" and "expressiveness".

5.1 Semantic Suitability

Smart contracts can be defined by the properties of contracting concepts: "Who," "Where," and "What." The Who-concept describes the parties engaged in the contracting process. The "Who" of an e-contract must consist of at least two parties involved in a legally valid contract. In addition, a mediator is often included in the contract establishment. Parties specify the rights and obligations in a contract so that if one party uses their rights, there is a corresponding obligation on the other party. The context of contracts is defined in the content that affects the actors' roles, exchange values, and contracting processes. The Where-concept describes the legal and business context for the contract establishment. The legal provisions are specified in the contract to resolve disputes, while the business context has an essential role in setting the requirements for the contracting processes [6]. Finally, the What-concept describes the exchange value, and its provision is described for every contractual party being services, products, and financial rewards. Also, service descriptions, e.g., service type, role, and so on, are specified in a contract as exchange values. An exchange-value provision is required to describe the process flow in a successful value exchange [77].

5.2 Workflow Suitability

These patterns describe the fundamental requirements that arise on a recurring basis during business-process modeling. Smart contracts follow the contracting processes and thus, we identify the patterns required for the designing of SCL in Table 12. The said patterns include control-flow, data-flow, resource-flow, exception-handling, and event logs. The control-flow patterns [87], such as XOR and AND, are used to depict the relation of activity-flow in the designing of imperative models. Whereas the data-flow patterns describe how data are represented and utilized to ensure transparency in business-processes [42] and also characterize the interaction of data elements among processes. Currently, most existing languages focus on the control- and data-flow, and, unfortunately, resource perspectives are not considered. Resources such as humans, machines, and their roles must be correct, which affects the simulation of the in-house process of service consumer and provider of business collaboration [88]. The deviation from the normal execution arising during a business process is called exceptions since the unanticipated events such as integer overflow, call stack, re-entrancy, and so on, are difficult to characterize. Therefore, an exception handler resolves the effect of such events as they are detected. An event log is a collection of multiple records and is represented as a sequence of events carried out in a single execution process. The aim of an event log is to uncover useful information relevant to the business processes by applying several techniques, such as data- and process mining [96].

5.3 Expressiveness

Expressiveness properties pertain to formal mathematical correctness of smart-contract code [77]. Several verification tools such as Oyente, Securify, SmartCheck, and others are implemented to detect the unintended behavior of smart contracts [71]. With this, the parties can verify the functionality of smart contracts before posting to blockchain platforms. Several formal methods are required to verify contracts' semantics, including static analysis, model checking, and formal semantics, and so on. For instance, Securify Tool [71] is a security analyzer for Ethereum contracts that symbolically analyzes the contract behavior by extracting exact semantic information from the code. Further, we discover the formal representation of smart contracts that comprise the temporal- and structural properties [99]. Events, obligations, and rights of contracts are the temporal properties, because their state of execution depends on time [94]. Each event has a specified date and time for taking action, e.g., pay, or time to deliver. Similarly, obligations have a start date, due date, and discharge date. Rights also have a specified date/time and become activated when events, or dates occur. Structural properties pertain to the functional behavior of smart contracts. To validate the functional correctness of the smart contract, Liu et al. [63] propose a formal verification method based on **Color Petri Nets (CPN)**. CPN[14] is a language for specification, simulation, and design of systems. As an alternative, statechart is used to define the structural properties and constraints on the sequence of states in a state machine [99].

We identify the critical SCL properties to make smart contracts legally binding, i.e., "suitability" and "expressiveness." Furthermore, we enumerate the said properties and their associated studies in Table 12. In the next section, we discover these properties in existing SCLs.

6 EVALUATION OF SCLS' SUITABILITY AND EXPRESSIVENESS

We evaluate the existing SCLs based on the contractual enabling properties identified from RQ2 as per Section 5. To evaluate the SCLs, we score them with "+" or "-" operators for each of the 10 properties. The former indicates that the SCL has the particular property, whereas the latter indicates that the property does not exist in the corresponding SCL. In case we are not able to identify said properties of an SCL, we indicate the same using *n/a.* The final results of the evaluation are presented in Table 13.

- *RQ3*: What are the obstacles in existing SCLs that restrict the achievement of business-contractual objectives?

In this RQ, we aim to evaluate the suitability and expressiveness of the SCLs identified from *RQ1* as per Section 4. We allude to earlier that not all SCLs are designed the same; and thus, while examining the supporting studies in Section 5, we discover that many SCLs are not described as having all the identified suitability- and expressiveness properties (see Table 12). Thus, for this RQ, we reexamine the 45 selected studies again, together with the 28 supporting studies, and finally score the 45 SCLs as shown in Table 13. Additionally, we detail our reasoning behind the scores allocated to each of the properties of the SCLs. By doing so, we aim to identify SCLs that can potentially be used for drafting legally-binding smart contracts in DAO collaborations.

[14]CPN | Home Page.

Table 13. Evaluation of SCLs Pertaining to Business-contractual Aspects

SCL Types	SCLs	Suitability*								Express-iveness* (*viz. Formal Verification)*	
		Semantic			Workflow						
		WO	WR	WT	CF	DF	RF	ExH	ElI	TC	SC
Domain Specific	Solidity (GL13)	+	-	-	+	-	-	+	+	-	+
	Vyper (GL12)	+	-	-	+	+	-	+	+	+	-
	Idris (SS14)	+	-	-	+	+	-	-	-	-	+
	Flint (SS13)	+	-	-	+	+	+	+	-	-	-
	Formality (GL14)	+	-	-	+	+	+	-	-	+	-
	Pyramid (GL15)	+	-	-	+	+	-	-	-	-	-
	Findel (SS5)	+	+	+	-	+	+	-	-	+	+
	Michelson (GL9)	+	-	-	+	+	+	-	-	-	+
	Plutus-Core (GL10)	+	-	-	-	-	+	-	-	-	+
	Rholang (GL11)	+	-	-	+	+	+	+	+	+	+
	Marlowe (SS16)	+	+	-	-	+	-	-	-	-	+
	Pact (GL2)	+	-	-	-	+	-	-	-	+	-
	Sophia (GL24)	+	-	-	+	+	-	+	+	-	-
	Fift (GL25)	+	-	-	+	+	-	+	+	-	-
	Ivy (GL5)	+	-	-	-	+	+	-	-	-	-
	Typecoin (SS17)	+	-	-	-	+	+	-	-	-	-
Formal Verifi-cation	Lolisa (GL26)	+	-	-	+	+	-	+	-	+	+
	Bamboo (GL6)	+	-	-	-	+	-	-	-	+	+
	Mutan (GL19)	+	-	-	+	+	-	-	-	+	+
	Script (SS18)	+	-	-	-	-	-	-	-	-	+
	Move (GL21)	+	+	+	-	+	+	-	-	+	+
	Obsidian (GL27)	+	-	-	+	+	+	-	-	-	-
	IELE (GL23)	+	-	-	+	+	+	-	-	-	+
	Fi (GL7)	+	-	-	+	+	-	-	+	-	+
	Scilla (SS9)	+	-	-	-	+	-	-	-	-	+
	BitML (SS3)	+	-	-	-	-	-	+	-	-	+
	Simplicity (SS1)	+	-	-	-	+	+	-	-	+	-
	Liquidity (GL8)	+	-	-	+	+	-	-	-	+	+
	F-Sol (GL17)	+	-	-	-	+	-	-	-	-	+
	FSolidM (SS15)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	BALZaC (GL3)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
User Friend-liness	LLL (GL16)	+	-	-	+	+	-	-	-	+	+
	Babbage (GL18)	+	-	-	-	+	-	-	-	-	+
	RIDE (GL22)	+	-	-	-	-	+	+	-	-	-
	DSL4SC (SS2)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	SmaCoNat (SS10)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

(Continued)

Table 13. Continued

SCL Types	SCLs	Suitability*								Express-iveness* (*viz. Formal Verification)*	
		Semantic			Workflow						
		WO	WR	WT	CF	DF	RF	ExH	ElI	TC	SC
Legally Enforce-able	ADICO (SS7)	+	+	+	-	+	-	+	-	-	+
	ErgoScript (GL20)	+	+	+	-	-	+	-	-	-	+
	SPESC (SS49)	+	+	+	-	-	+	-	-	-	-
	Reaction-Rule ML (SS1)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	Commit-Rule ML (SS8)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Business Process	DAML (GL4)	+	+	+	-	+	+	-	+	-	-
	eSML (SS4)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	BCRL (SS6)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
	QSCL (GL1)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a

* *WO* [Who-concept] | *WR* [Where-concept] | *WT* [What-concept].
* *CF* [Control-flow] | *DF* [Data-flow] | *RF* [Resource-flow] | *ExH* [Exception-handling] | *ElI* [Event log imperfection].
* *TC* [Temporal-constraint] |*SC* [Structural-constraint].

Our investigation shows that Solidity expresses the *Who-concept* by including the contracting party's address using the *address-owner* variable. Furthermore, control- and data-flow properties are defined due to the imperative paradigm of Solidity. Since data are presented in states encoded with a highly secured cryptography technique stored at a specific blockchain address, it is not easy for non-domain users to understand the smart contracts' dataflow. The variable- and data types are not considered to support Where- and What-concept, because Solidity focuses on manipulating low-level blockchain in java-script style. The use of modifiers *onlyBy()* enables the ownership of resources, and Solidity expresses exception handling only for external function- and contract-creation calls in the form of *try/catch* statements. Solidity employs logs to implement events and a contract accesses the log-data after being created. Still, recent attacks show a lack of formal verification in Solidity; while the *uint start-data*, *uint end-data* variables and modifiers such as *checkTime()* and *onlyOnce()*, satisfy the temporal constraint. The other domain-specific languages, namely Idris and Vyper support properties similar to Solidity, except for resource-flow. Vyper initializes a global variable a beneficiary by calling the public on the *public (address)* datatype and modifier *raise(reason: str)* returns the reason of exception. While, Idris expresses the *Who-concept* using modifier *mapping (address=>uint)* and latter includes *Raise* handlers to returns the exception. Unfortunately, Idris does not specify the enforcement of requirements among the processes due to the program's lack of data awareness.

Attacks exploit several unsafe patterns, such as call-to-the-unknown, gasless send, exception disorder, and others; resulting in loss of money, or other damages. A call to the unknown function is a primitive that invokes the function and transfer the digital asset. Exceptions arise in a situation when an execution runs out of gas. There is not a solution in imperative languages to handle such an exception.

Flint expresses the Who-concept by including the *address* variable, and the *type state* is used to satisfy control- and data-flow properties. Flint additionally supports resource flow by implementing *Asset type* that prevents a class of security flaws wherein the state of smart-contract inaccurately represents resources. Function modifiers such as *require* are used to check the precondition before entering the function body. Still, properties such as the *Where-* and What-concept are not expressed in Flint.

At the same time, the Flint language determines the resource consumption by implementing the Wei asset function. While in Solidity, Wei is defined as an integer value rather than a dedicated type, which allows the conversion between number and currency and thus, results in an inconsistent state in which the actual balance of a smart contract is incorrect. Formality and Pyramid scheme SCLs satisfy each property similar to Flint, except for structural constraints. Formality includes inductive data-types for expressing structural properties. In contrast, the Pyramid scheme expresses structural properties by combining formal IR semantics and dependently typed language.

Findel contract is defined as a tuple *(D, I, O)* where D is a description, I is an issuer, and O is the owner. Findel expresses the Who-concept by defining the owner and issuer's address, and describes the What-concept by defining the contract context in the description field. Formal semantics of Findel are introduced using a rigorously defined put-call parity algebra theorem and time-bound primitives to express temporal conditions. Yet, exception handling and event-log primitives are missing properties to achieve the objectives of SCLs. Michelson contains the *address* data-type that expresses the Who-concept properties. *Mutez* types allow restricting the manipulation of resources that reveal the *resource-flow* properties. Furthermore, using the timestamp Michelson code satisfies the *temporal-constraint* properties. The Now() and Add() functions allow the retrieval of current timestamps of events. Other Tezos blockchain-based languages such as Liquidity and Fi satisfy the underlying properties of Michelson, also with qualifying the structural properties.

The Rholang syntax and semantics are similar to rho calculus, a reflective higher-order variant of pi-calculus. Resource flow is expressed in Rholang by implementing the *phlogiston metric*, which is identical to Ethereum gas. Rholang considers patterns for the storage and retrieval of information via channels. Unlike other SCLs, Marlowe expresses the *Where-concept* by implementing the type of contract in a Haskell datatype. In addition, several of the contracts have timeouts that establish their behavior as well. Marlowe implements the *step function* that operates on each constructor of contract type and thus, satisfies the temporal constraints. Pact SCL (GL2) employs the *keyset* variable to hold the parties' addresses and favors a declarative approach over complex control-flow, making bugs more challenging to write and easier to detect. Sophia expresses properties such as the *Who-concept*, *control-flow*, *data-flow*, *eventlog*, and Exception handling. Sophia employs the *address* literals to define a contract, or the parties' address. A contract declares a datatype *event* to use events that are logged using the *chain.event function*. Contracts can fail with an (uncatchable) exception using the built-in function: abort (reason : string). Fift uses the *abort* inbuilt function to throw an exception with an error message. The reason for the development of Ivy (GL5) and Typecoin language is to write scripting code for cryptocurrency and is less focused on supporting smart-contract semantics.

Lolisa includes Solidity features such as mapping, modifier, contract, and address types. The intention behind the development of Lolisa is to formalize the Solidity programming languages. Bamboo, Move, and Obsidian is mainly intended to support resource-flow. Bamboo expresses the *Who-concept* using variable *(address=> uint256)* to identify the parties address and also expel structural properties by implementing the reentrancy-guard for calling recursive functions. Move's type-system ensures that resources never be copied, duplicated, and lost. Failing to move a resource triggers a bytecode verification error, e.g., by deleting the line that contains movecoin. Obsidian uses a type-system to ensure that assets are manipulated correctly and employs linear types to allow a safe manipulation of objects. Additionally, Obsidian expresses the *control-* and *data-flow* properties by implementing the polymorphic linked list that is a container to store the transactions. The current practice of representing Bitcoin contracts as cryptographic protocols expresses the *Who-concept* and *control-flow* using the opcodes that are a list of scripts.

BitML expresses the *temporal constraints* by choosing a secret and revealing it using time constraint 't', and introduces the symbolic and computation model to describe structural properties. Simplicity seeks to express resource flow using a formal semantics that allows for "fast" (linear time) static analysis of resource consumption. Scilla expresses the resource-flow using variable *mapping (address => uint) assets* and defines the *address* variable to express the *Who-concept* properties. Integrating the Scilla into the proof assistant of *Coq* enables it to reason about the properties of the security and temporal-constraints. IELE expresses the *control-flow* in which the body of the function contains code organized into labeled blocks. When a branch instruction is encountered, the execution falls through from the last instruction of a block to the first of the next one, or jumps to the beginning of a specific block.

It is important to point out here that several foundations such as ADICO, ErgoScript, CommitRuleML, ReactionRuleML, SPECS, BCRL, and eSML are proposed in academic papers that have unfortunately not spawned any further implementation or research efforts. And thus, the Supporting Studies for said SCLs have not been published in the literature yet. Hence, we have presented the properties of said SCLs with the "n/a" (i.e., not-applicable) symbol in Table 13.

7 DISCUSSION

Drawing from the results of the RQs, it becomes clear that none of the state-of-the-art SCLs identified and analyzed in this SLR in their current forms without external support such as smart-contract templates, or frameworks, are capable of supporting a creation of legally binding smart contracts for collaborations in DAOs. Although there exist alternative solutions for designing semantically correct [25] and formally verifiable [63] contracts, we discover that to assure the robustness of smart contracts, future SCLs should be designed systematically from the top-down based on sound principles. As per our work, we provide insights into how this could be achieved.

To identify potential SCLs that allow for drafting legally-binding smart contracts, we check if the state-of-the-art SCLs comprise 10 properties described in Section 5 that are categorized as semantic suitability, workflow suitability, and expressiveness. Note, we describe the meaning of each of these categorizations from a legal perspective to explore why these categories and their properties are critical in making smart contracts acceptable in the court of law. As explained in Section 5, semantic suitability encompasses the Who-Where-What properties that describe the parties involved in a contract. Without a semantically rich SCL that has the necessary vocabulary required to describe the who-where-what of the contract, the contract cannot even be entertained by the law [49], thereby making said properties all-important. Workflow suitability properties, however, are crucial for understanding which partner/partners are not fulfilling their specific obligations [104]. As their names suggest, these properties describe how the flow of control, data and resources should occur under the contract and in-case an anomaly occurs, the court of law can then intervene to resolve potential conflicts accordingly. Finally, formal verifiability, or expressiveness is critical for making contracts legally binding, as these properties ensure that contracts being computer programs, are encoded correctly based on the legal intention of the parties as is, commonly defined in a natural-language contract [49, 67].

To summarize the findings from RQs, we find that while most SCLs supports some suitability- and expressiveness properties; however, none of the current state-of-the-art SCLs have all 10 of said properties (see Table 5). While most SCLs do have some semantic- (e.g., Who-concept) and workflow properties (e.g., control-flow and ddata-flow); nearly none of the studied SCLs have adopted all workflow suitability properties (except Rholang (GL11)), whereas in the particular case of legally enforceable and business process SCLs, the studied languages do fulfill all three (who-where-what) semantic suitability properties. Still, business rules and policies are an essential factor for a smart contract and this is equivalent to the formulation of obligations and rights in traditional contracts. Furthermore, a smart contract must uniquely identify who the collaborating parties are and currently, user-friendly SCLs do not have the syntax for supporting this property. Besides that, the unique identification of parties specifies the required competencies, authorization, and so on, that are a part of the resource perspective of a process for which SCLs also lack concepts and properties in the existing syntax definitions. With respect to the expressiveness properties, we find that most SCLs either have temporal constraints, or structural constraints, but only a subset of SCLs (such as Rholang (GL11), Lolisa (GL26), LLL (GL16), and others) have both said properties. This is critical as having these properties makes these SCLs less prone to security vulnerabilities [99]. Thereby, the question arises of how to develop an SCL that incorporates the contractual semantics to achieve legally-enforceable smart contracts. At the same time, SCL is capable of defending against hacks and attacks (such as the DAO attack and the Parity Wallet hack) that occur due to security vulnerabilities caused by a lack of time constraints, predictable states, exception handling, and other factors.

The challenges mentioned above need to be overcome to achieve the full potential of smart contracts for DAOs (and other kinds of business collaborations). To this end, based on current literature, we find that these gaps in the state of the art can be addressed in two ways, first, by utilizing modularization- and simplification techniques to develop graphical SCLs for writing semantically correct smart-contract codes (as shown in Reference [105]). Codes that are known to be correct, less susceptible to errors, and are based on predefined procedures/best practices can be broken down into modules and reused in novel solutions. The second approach for addressing the previously mentioned gaps is to develop novel SCLs from the top-down based on sound principles and with scientific menthods while learning from the insights provided in published literature. In our future work, we aim to focus on the latter in that we would like to propose a semantically rich SCL that is process-driven and formally verifiable, as shown in the Figure 2. Building on Norta et al.'s eSML 1.0 [77], we would like to develop eSML 2.0, which will be rich in legal semantic vocabulary, will be robust enough to tackle potential temporal and structural vulnerabilities, and will be deployable over real-world blockchains [39]. To achieve this, we have already conducted qualitative interviews with legal professionals (i.e., experts in business law) and created a novel ontology that is semantically rich enough to describe all business collaboration processes (using the Protégé tool[15] and HermiT reasoner.[16] Next, we mapped the created legal ontology with eSML 1.0 and thus, created eSML 2.0. As part of our ongoing research, we are currently looking into building a compiler for eSML 2.0 grammar using lexer and parser generators (such as ANTLR [81] or Xtext [108]). Once we have developed a functional languag from eSML 2.0, programmers could choose either to draft smart contracts in the said functional languag (i.e., high-level language) directly, or use Extensible Markup Language (XML) templates, which could then be translated into the blockchain's high-level language.

8 THREATS TO VALIDITY

The study selection in the SLR was based on the search strategy, and includes the selection of search terms, literature resources, and the search process. As described in Section 3, we identify the search terms by examining both the research questions and initial literature search (presented in Section 3.1.1). By considering previous literature, we are able to identify the keywords and their commonly used synonyms (and variants). Followed by this, we generate the search query

[15]Protege | Home Page.
[16]HermiT | Data and Knowledge Group.

(presented in Table 2) and run the search queries on well-renowned scholarly databases (including Scopus and Web of Science). Using this methodology, we are able to identify relevant white literature (books, journal articles, proceedings, and workshop papers). To identify additional white literature and relevant grey literature (white papers, technical reports, and working drafts; and SCL documentation), we run the search query on Google Scholar and GitHub (respectively); in total, we are able to identify 616 pieces of literature (see Table 3).

To avoid biases in the selection process, we first study the titles and abstracts of all identified articles, and then shortlist the studies that fulfill the inclusion- and exclusion criteria. To validate the accuracy of the study-selection phase, three of the co-authors independently re-examine the initially identify 130 articles against the study selection criteria; and thus assure that only the articles relevant to the research questions are selected as primary studies. Finally, we evaluate the primary studies based on the quality assessment criteria, and thereby, we were able to identify only those studies that are most relevant for answering our research questions. Throughout the process of the SLR, all conflicts including the formulation of the inclusion-, exclusion- and quality-assessment criteria, and the establishment of the selected- and supporting studies are resolved through discussions and deliberations; and are therefore finalized based on the consensus of all the authors.

After the quality-assessment phase, we discover that the same authors publish the selected studies SS22, SS24, and SS26. Still, after independently studying the said articles, we agree that each of the articles present different contributions and are therefore appropriate as selected studies. Similarly, studies SS33 and SS34 are also found to be quite similar. Still, after a detailed reading of the articles, we discover that both articles describe formal-verification techniques but present different contexts; hence, we decide to include both articles as supporting studies.

To assure reproducibility of the SLR's search- and study-selection process, especially given that we include articles published as grey literature; we archive the GitHub pages and other web resources using the Internet Archive's Wayback Machine.[17] This step is critical to our research methodology, as web resources (especially GitHub pages and SCL documentation) can often change over time, since developers/researchers keep adding new features to their artifacts (viz. SCLs). Archiving the web resources in their states (as they are when the SLR is conducted) assures that readers of the SLR would have access to a copy of the resources even after the web pages change, or deleted.

9 CONCLUSION

Blockchains and smart contracts are emerging technologies that enable a plethora of novel applications in diverse domains. In our work, we focus on one such field of research; namely, the development of e-contracts for DAO collaborations. Although smart contracts have gained a lot of attention from both academia and industry in recent years, available literature has not yet led to the development of legally enforceable smart contracts and SCLs. To address this research gap, we investigate 45 state-of-the-art smart-contract languages designed with a focus on business-collaboration processes. We identify said SCLs by conducting an exhaustive SLR of 616 articles published as white- and grey-literature between 2015 and 2019. Based on a carefully framed inclusion-, exclusion- and quality-assessment criteria, we are able to identify 45 selected studies and 28 supporting studies. Drawing from said studies, we identify 10 critical properties categorized as semantic suitability, workflow suitability and expressive, which can render smart contracts legally enforceable. We discover that none of the identified state-of-the-art SCLs have all of the suitability- and expressiveness properties. Finally, based on these findings, we propose our future research

[17]Wayback Machine | Homepage.

direction for developing legally enforceable SCLs. We propose that legally enforceable SCLs could be achieved through the development of an ontology for contractual business semantics. Mapping said semantic ontology onto the eSML 1.0's contractual choreography language, could lead to the development of a novel functional language for smart contracting DAO collaborations. As future work we aim to render business collaborations more flexible, effective and efficient with legally enforceable and practically usable SCLs.

REFERENCES

[1] Ebizimoh Abodei, Alex Norta, Irene Azogu, Chibuzor Udokwu, and Dirk Draheim. 2019. Blockchain technology for enabling transparent and traceable government collaboration in public project processes of developing economies. In *Lecture Notes in Computer Science*. Springer International Publishing, 464–475. DOI:https://doi.org/10.1007/978-3-030-29374-1_38

[2] Temofe Isaac Akaba, Alex Norta, Chibuzor Udokwu, and Dirk Draheim. 2020. A framework for the adoption of blockchain-based e-procurement systems in the public sector. In *Lecture Notes in Computer Science*. Springer International Publishing, 3–14. DOI:https://doi.org/10.1007/978-3-030-44999-5_1

[3] Maher Alharby and Aad van Moorsel. 2017. Blockchain based smart contracts: A systematic mapping study. In *Computer Science & Information Technology (CS & IT)*. Academy & Industry Research Collaboration Center (AIRCC). 125–140. DOI:https://doi.org/10.5121/csit.2017.71011

[4] Maher Alharby and Aad van Moorsel. 2017. A systematic mapping study on current research topics in smart contracts. *Int. J. Comput. Sci. Inf. Technol.* 9, 5 (Oct. 2017), 151–164. DOI:https://doi.org/10.5121/ijcsit.2017.9511

[5] Stephen Andrews. 2019. Introduction—Learn Fi. Retrieved from https://learn.fi-code.com/.

[6] S. Angelov. 2006. *Foundations of B2B Electronic Contracting*. Ph.D. Dissertation. Industrial Engineering & Innovation Sciences. DOI:https://doi.org/10.6100/IR600020

[7] Tara Astigarraga, Xiaoyan Chen, Yaoliang Chen, Jingxiao Gu, Richard Hull, Limei Jiao, Yuliang Li, and Petr Novotny. 2018. Empowering business-level blockchain users with a rules framework for smart contracts. In *Service-Oriented Computing*. Springer International Publishing, Cham, 111–128. DOI:https://doi.org/10.1007/978-3-030-03596-9_8

[8] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. A survey of attacks on Ethereum smart contracts (SoK). In *Lecture Notes in Computer Science*. Springer, Berlin, 164–186. DOI:https://doi.org/10.1007/978-3-662-54455-6_8

[9] Nicola Atzei, Massimo Bartoletti, Stefano Lande, and Roberto Zunino. 2018. A formal model of bitcoin transactions. In *Financial Cryptography and Data Security*, Sarah Meiklejohn and Kazue Sako (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 541–560. DOI:https://doi.org/10.1007/978-3-662-58387-6_29

[10] Irene Azogu, Alex Norta, Ingrid Papper, Justin Longo, and Dirk Draheim. 2019. A framework for the adoption of blockchain technology in healthcare information management systems. In *Proceedings of the 12th International Conference on Theory and Practice of Electronic Governance - ICEGOV2019*. ACM Press, 310–316. DOI:https://doi.org/10.1145/3326365.3326405

[11] Massimo Bartoletti and Roberto Zunino. 2018. BitML: A calculus for bitcoin smart contracts. In *Proceedings of the ACM Conference on Computer and Communications Security*. Association for Computing Machinery, 83–100. DOI:https://doi.org/10.1145/3243734.3243795

[12] A. Begicheva and I. Smagin. 2019. *Ride: A Smart Contract Language for Waves*. Technical Report.

[13] S. Farrell, H. Machin, and R. Hinchliffe. 2017. 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 (Wien, Uncitral, Ebook 2017)*. 95–104.

[14] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. 2016. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security (PLAS'16)*. Association for Computing Machinery, New York, NY, 91–96. DOI:https://doi.org/10.1145/2993600.2993611

[15] Alex Biryukov, Dmitry Khovratovich, and Sergei Tikhomirov. 2017. Findel: secure derivative contracts for ethereum. In *Financial Cryptography and Data Security*. Springer International Publishing, 453–467. https://doi.org/10.1007/978-3-319-70278-0_28

[16] Sam Blackshear, Evan Cheng, David L. Dill, Victor Gao, Ben Maurer, Todd Nowacki, Alistair Pott, Shaz Qadeer, Dario Russi, Stephane Sezer, Tim Zakian, and Runtian Zhou. 2019. *Move: A Language with Programmable Resources*. Technical Report.

[17] Willi Brammertz and Allan I. Mendelowitz. 2018. From digital currencies to digital finance: The case for a smart financial contract standard. *J. Risk Financ.* 19, 1 (Jan. 2018), 76–92. DOI:https://doi.org/10.1108/jrf-02-2017-0025

[18] Mario Bravetti and Gianluigi Zavattaro. 2007. Towards a unifying theory for choreography conformance and contract compliance. In *Software Composition*, Markus Lumpe and Wim Vanderperren (Eds.). Springer, Berlin, 34–50. DOI: https://doi.org/10.1007/978-3-540-77351-1_4
[19] Michael Burg. 2017. Write Your Next Ethereum Contract in Pyramid Scheme. Retrieved October 30, 2018 from http://www.michaelburge.us/2017/11/28/write-your-next-ethereum-contract-in-pyramid-scheme.html.
[20] Vitalik Buterin. 2018. Vyper Documentation. Retrieved October 30, 2018 from https://media.readthedocs.org/pdf/viper/latest/viper.pdf.
[21] Cardano. 2019. Introduction—Cardano. Retrieved October 30, 2018 from https://cardanodocs.com/technical/plutus/introduction/.
[22] Fran Casino, Thomas K Dasaklis, and Constantinos Patsakis. 2019. A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telemat. Informat.* 36 (2019), 55–81. DOI: https://doi.org/10.1016/j.tele.2018.11.006
[23] Usman W. Chohan. 2017. The decentralized autonomous organization and governance issues. *SSRN Electr. J.* (2017), 1–7. DOI: https://doi.org/10.2139/ssrn.3082055
[24] Christian. 2017. Babbage—A Mechanical Smart Contract Language. Retrieved October 30, 2018 from https://medium.com/@chriseth/babbage-a-mechanical-smart-contract-language-5c8329ec5a0e.
[25] Christopher D. Clack. 2018. Smart contract templates: Legal semantics and code validation. *J. Digit. Bank.* 2, 4 (2018), 338–352.
[26] Michael Coblenz, Reed Oei, Tyler Etzel, Paulette Koronkevich, Miles Baker, Yannick Bloem, Brad A. Myers, Joshua Sunshine, and Jonathan Aldrich. 2020. Obsidian: Typestate and assets for safer blockchain programming. *ACM Trans. Program. Lang. Syst.* 42, 3, Article 14 (December 2020), 82 pages. DOI: https://doi.org/10.1145/3417516
[27] Karl Crary and Michael J. Sullivan. 2015. Peer-to-peer affine commitment using Bitcoin. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'15)*, Vol. 2015. Association for Computing Machinery, 479–488. DOI: https://doi.org/10.1145/2737924.2737997
[28] Patrick Dai. 2017. *Smart-Contract Value-Transfer Protocols on a Distributed Mobile Application Platform.* Technical Report. Singapore.
[29] Primavera De Filippi and Samer Hassan. 2016. Blockchain technology as a regulatory technology: From code is law to law is code. *First Monday* 21, 12 (2016). DOI: https://doi.org/10.5210/fm.v21i12.7113
[30] Joost De Kruijff and Hans Weigand. 2018. An introduction to commitment based smart contracts using ReactionRuleML. In *Proceedings of the CEUR Workshop*, Proper E. Gordijn J. (Ed.), Vol. 2239. CEUR-WS, 149–157.
[31] Joost De Kruijff and Hans Weigand. 2019. Introducing commitRuleML for smart contracts. In *Proceedings of the CEUR Workshop*, H. Weigand, P. Johannesson, and B. Andersson (Eds.), Vol. 2383. CEUR-WS.
[32] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí. 2020. A fair protocol for data trading based on Bitcoin transactions. *Fut. Gener. Comput. Syst.* 107 (2020), 832–840. DOI: https://doi.org/10.1016/j.future.2017.08.021
[33] Ergo Developers. 2019. *ErgoScript, a Cryptocurrency Scripting Language Supporting Noninteractive Zero-Knowledge Proofs.* Technical Report. 1–22 pages.
[34] DigitalAsset. 2019. *DAMLScript SDK Documentation.* Technical Report. Retrieved December 2, 2020 from https://docs.daml.com/.
[35] Dirk Draheim. 2021. Blockchains from an e-governance perspective: Potential and challenges. In *Proceedings of the 7th International Conference on Electronic Governance and Open Society—Challenges in Eurasia (EGOSE'20)*,Communications in Computer and Information Science. Springer, 11–13. https://doi.org/10.1007/978-3-030-67238-6
[36] Quinn DuPont. 2017. Experiments in algorithmic governance: A history and ethnography of "The DAO," a failed decentralized autonomous organization. In *Bitcoin and Beyond: Cryptocurrencies, Blockchains and Global Governance.* Routledge, 157–177. DOI: https://doi.org/10.4324/9781315211909-8
[37] Nikolai Durov. 2019. *Fift: A Brief Introduction.* Technical Report.
[38] Vimal Dwivedi, Vipin Deval, Abhishek Dixit, and Alex Norta. 2019. Formal-verification of smart-contract languages: A survey. In *Advances in Computing and Data Sciences.* Springer Singapore, 738–747. DOI: https://doi.org/10.1007/978-981-13-9942-8_68
[39] Vimal Kumar Dwivedi and Alex Norta. 2018. A legally relevant socio-technical language development for smart contracts. In *Proceedings of the 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS*W'18).* Institute of Electrical and Electronics Engineers Inc., 11–13. DOI: https://doi.org/10.1109/FAS-W.2018.00016
[40] Ben Edgington. 2017. *LLL Compiler Documentation Documentation Release 0.1.* Technical Report.
[41] Al K. Firas et al.2017. Trust in smart contracts is a process, as well. In *Financial Cryptography and Data Security.* Springer International, 510–519. DOI: https://doi.org/10.1007/978-3-319-70278-0_32

[42] R. Nick et al.2005. Workflow resource patterns: Identification, representation and tool support. In *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*. Springer International Publishing, 216–232. DOI:https://doi.org/10.1007/11431855_16

[43] Ethereum. 2014. *Solidity Documentation*. Technical Report.

[44] C. K. Frantz and M. Nowostawski. 2016. From institutions to code: Towards automated generation of smart contracts. In *Proceedings of the 2016 IEEE 1st International Workshops on Foundations and Applications of Self* Systems (FAS*W'16)*. 210–215. DOI:https://doi.org/10.1109/FAS-W.2016.53

[45] Mark Giancaspro. 2017. Is a "smart contract' really a smart idea? Insights from a legal perspective. *Comput. Law Secur. Rev.* 33, 6 (2017), 825–835. DOI:https://doi.org/10.1016/j.clsr.2017.05.007

[46] Jean-Yves Girard. 1987. Linear logic. *Theor. Comput. Sci.* 50, 1 (1987), 1–101. DOI:https://doi.org/10.1016/0304-3975(87)90045-4

[47] Jake Goldenfein and Andrea Leiter. 2018. Legal engineering on the blockchain: 'Smart contracts' as legal conduct. *Law Crit.* 29, 2 (May 2018), 141–149. DOI:https://doi.org/10.1007/s10978-018-9224-0

[48] David Golumbia. 2016. *The Politics of Bitcoin: Software as Right-wing Extremism*. University of Minnesota Press.

[49] Guido Governatori, Florian Idelberger, Zoran Milosevic, Regis Riveret, Giovanni Sartor, and Xiwei Xu. 2018. On legal contracts, imperative and declarative smart contracts, and blockchain systems. *Artif. Intell. Law* 26, 4 (2018), 377–409. DOI:https://doi.org/10.1007/s10506-018-9223-3

[50] Ilya Grishchenko, Matteo Maffei, and Clara Schneidewind. 2018. Foundations and tools for the static analysis of Ethereum smart contracts. In *Computer Aided Verification*, Hana Chockler and Georg Weissenbacher (Eds.). Springer International, Cham, 51–78. DOI:https://doi.org/10.1007/978-3-319-96145-3_4

[51] Purva Grover, Arpan Kumar Kar, and P. Vigneswara Ilavarasan. 2018. Blockchain for businesses: A systematic literature review. In *Conference on e-Business, e-Services and e-Society*. Springer, 325–336. DOI:https://doi.org/10.1007/978-3-030-02131-3_29

[52] Dominik Harz and William Knottenbelt. 2018. Towards safer smart contracts: A survey of languages and verification methods. arXiv:1809.09805. Retrieved from https://arxiv.org/abs/1809.09805.

[53] Florian Hawlitschek, Benedikt Notheisen, and Timm Teubner. 2018. The limits of trust-free systems: A literature review on blockchain technology and trust in the sharing economy. *Electr. Commerce Res. Appl.* 29 (2018), 50–63. DOI:https://doi.org/10.1016/j.elerap.2018.03.005

[54] X. He, B. Qin, Y. Zhu, X. Chen, and Y. Liu. 2018. SPESC: A specification language for smart contracts. In *Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC'18)*, Vol. 01. 132–137. DOI:https://doi.org/10.1109/COMPSAC.2018.00025

[55] IOHK. 2019. plutus/extended-utxo-spec at master · input-output-hk/plutus · GitHub. Retrieved October 30, 2020 from https://github.com/input-output-hk/plutus/tree/master/extended-utxo-spec.

[56] Mubashar Iqbal and Raimundas Matulevičius. 2019. Blockchain-based application security risks: A systematic literature review. In *Proceedings of the International Conference on Advanced Information Systems Engineering*. Springer, 176–188. DOI:https://doi.org/10.1007/978-3-030-20948-3_16

[57] Christoph Jentzsch. 2016. Decentralized autonomous organization to automate governance. *White Paper, November* (2016).

[58] Theodoros Kasampalis. 2018. *IELE: An Intermediate-Level Blockchain Language Designed and Implemented Using Formal Semantics*. Technical Report.

[59] Barbara Kitchenham and Stuart Charters. 2007. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report. Technical Report, Ver. 2.3 EBSE Technical Report. EBSE.

[60] Markus Knecht. 2018. Mandala: A smart contract programming language. arXiv:1911.11376. Retrieved from https://arxiv.org/abs/1911.11376.

[61] Pablo Lamela Seijas and Simon Thompson. 2018. Marlowe: Financial contracts on blockchain. In *Proceedings of the International Symposium on Leveraging Applications of Formal Methods*, Vol. 11247 LNCS. 356–375. DOI:https://doi.org/10.1007/978-3-030-03427-6_27

[62] Nino Lazuashvili, Alex Norta, and Dirk Draheim. 2019. Integration of blockchain technology into a land registration system for immutable traceability: A casestudy of Georgia. Springer International Publishing, Cham, 219–233. DOI:https://doi.org/10.1007/978-3-030-30429-4_15

[63] Zhentian Liu and Jing Liu. 2019. Formal verification of blockchain smart contract based on colored Petri net models. IEEE, 555–560. DOI:https://doi.org/10.1109/compsac.2019.10265

[64] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, Ingo Weber, and Alexander Ponomarev. 2019. Caterpillar: A business process execution engine on the Ethereum blockchain. *Softw.: Pract. Exp.* 49, 7 (2019), 1162–1193. DOI:https://doi.org/10.1002/spe.2702

[65] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*. Association for Computing Machinery, New York, NY, 254–269. DOI:https://doi.org/10.1145/2976749.2978309

[66] Daniel Macrinici, Cristian Cartofeanu, and Shang Gao. 2018. Smart contract applications within blockchain technology: A systematic mapping study. *Telemat. Informat.* 35, 8 (2018), 2337–2354. DOI: https://doi.org/10.1016/j.tele.2018.10.004

[67] D. Magazzeni, P. McBurney, and W. Nash. 2017. Validation and verification of smart contracts: A research agenda. *Computer* 50, 9 (2017), 50–57. DOI: https://doi.org/10.1109/MC.2017.3571045

[68] MaiaVictor. 2019. Formality Documentation Release 0.3.157. Retrieved November 5, 2020 from https://github.com/MaiaVictor/Formality.

[69] Anastasia Mavridou and Aron Laszka. 2018. Tool demonstration: FSolidM for designing secure ethereum smart contracts. In *Proceedings of the International Conference on Principles of Security and Trust*, Lecture Notes in Computer Science, Vol. 10804. Springer, 270–277. DOI: https://doi.org/10.1007/978-3-319-89722-6_11 arxiv:1802.09949

[70] Lucius Gregory Meredith, Jack Pettersson, Gary Stephenson, Michael Stay, Kent Shikama, and Joseph Denman. 2018. Contracts, Composition, and Scaling: The Rholang specification 0.2. Retrieved May 1, 2021 from https://developer.rchain.coop/assets/rholang-spec-0.2.pdf.

[71] Andrew Miller, Zhicheng Cai, and Somesh Jha. 2018. Smart contracts and opportunities for formal methods. In *Proceedings of the International Symposium on Leveraging Applications of Formal Methods*. Springer, 280–299. DOI: https://doi.org/10.1007/978-3-030-03427-6_22

[72] Andrew Miller, Zhicheng Cai, and Somesh Jha. 2018. Smart contracts and opportunities for formal methods. In *Lecture Notes in Computer Science (Including Subseries LNAI and LNBI)*, Vol. 11247. Springer Verlag, 280–299. DOI: https://doi.org/10.1007/978-3-030-03427-6_22

[73] Malte Möser, Ittay Eyal, and Emin Gün Sirer. 2016. Bitcoin covenants. In *Financial Cryptography and Data Security*, Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff (Eds.). Springer, Berlin, 126–141. DOI: https://doi.org/10.1007/978-3-662-53357-4_9

[74] Satoshi Nakamoto. 2008. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Technical Report.

[75] Arvind Narayanan and Jeremy Clark. 2017. Bitcoin's academic pedigree. *Commun. ACM* 60, 12 (Nov. 2017), 36–45. DOI: https://doi.org/10.1145/3132259

[76] Alex Norta. 2015. Creation of smart-contracting collaborations for decentralized autonomous organizations. In *Lecture Notes in Business Information Processing*. Springer International Publishing, 3–17. DOI: https://doi.org/10.1007/978-3-319-21915-8_1

[77] Alex Norta, Lixin Ma, Yucong Duan, Addi Rull, Merit Kõlvart, and Kuldar Taveter. 2015. eContractual choreography-language properties towards cross-organizational business collaboration. *J. Internet Serv. Appl.* 6, 1 (Apr. 2015), 1–23. DOI: https://doi.org/10.1186/s13174-015-0023-7

[78] OCamlPro. 2018. Welcome to Liquidity's documentation!—Liquidity 1.05. Retrieved October 30, 2020 from https://www.liquidity-lang.org/doc/.

[79] Q. Pan and X. Koutsoukos. 2019. Building a blockchain simulation using the Idris programming language. In *Proceedings of the 2019 ACM Southeast Conference (ACMSE'19)*. Association for Computing Machinery, 190–193. DOI: https://doi.org/10.1145/3299815.3314456

[80] M. P. Papazoglou. 2003. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE'03)*.3–12. DOI: https://doi.org/10.1109/WISE.2003.1254461

[81] Terence Parr. 1998. Introduction to ANTLR. Retrieved November 25, 2020 from https://www.antlr.org/.

[82] Travis Patron. 2015. *The Bitcoin Revolution: An Internet of Money*. Travis Patron.

[83] Stuart Popejoy. 2016. The Pact Smart Contract Language. Retrieved from http://kadena. io/docs/Kadena-PactWhitepaper. pdf.

[84] Emanuel Regnath and Sebastian Steinhorst. 2018. SmaCoNat: Smart contracts in natural language. In *Forum on Specification and Design Languages*, Vol. 2018. IEEE Computer Society. DOI: https://doi.org/10.1109/FDL.2018.8524068

[85] Raine Revere. 2017. Functional-Solidity-Language. Retrieved October 30, 2020 from https://github.com/raineorshine/functional-solidity-language.

[86] Dan Robinson. 2017. Ivy: A Declarative Predicate Language for Smart Contracts Introduction: Two Blockchain Models. Retrieved November 2, 2020 from https://docs.ivylang.org/bitcoin/.

[87] Nick Russell, Arthur H. M. Ter Hofstede, Wil M. P. Van Der Aalst, and Nataliya Mulyar. 2006. Workflow control-flow patterns: A revised view. *BPM Center Report BPM-06-22*, 06–22.

[88] Nick Russell, Arthur H. M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst. 2005. Workflow data patterns: Identification, representation and tool support. 353–368. DOI: https://doi.org/10.1007/11568322_23

[89] Nick Russell, Wil Mp Van Der Aalst, and Arthur H. M. Ter Hofstede. 2016. *Workflow Patterns: The Definitive Guide*. MIT Press.

[90] Alexander Savelyev. 2017. Contract law 2.0: 'Smart' contracts as the beginning of the end of classic contract law. *Inf. Commun. Technol. Law* 26, 2 (2017), 116–134. DOI: https://doi.org/10.1080/13600834.2017.1301036

[91] Franklin Schrans, Susan Eisenbach, and Sophia Drossopoulou. 2018. Writing safe smart contracts in Flint. In *Proceedings of the Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming (Programming'18 Companion)*. ACM Press, New York, NY, 218–219. DOI: https://doi.org/10.1145/3191697.3213790

[92] Pablo Lamela Seijas, Simon J. Thompson, and Darryl McAdams. 2016. Scripting smart contracts for distributed ledger technology.*IACR Cryptology ePrint Archive* 2016 (2016), 1156.

[93] Ilya Sergey, Amrit Kumar, and Aquinas Hobor. 2018. *Scilla: A Smart Contract Intermediate-Level LAnguage.* arXiv:1801.00687 [cs.PL]

[94] Ilya Sergey, Vaivaswatha Nagaraj, Jacob Johannsen, Amrit Kumar, Anton Trunov, and Ken Chan Guan Hao. 2019. Safer smart contract programming with Scilla. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 185 (Oct. 2019), 30 pages. DOI: https://doi.org/10.1145/3360611

[95] M. Singh and S. Kim. 2019. Blockchain technology for decentralized autonomous organizations. *Advances in Computers* 115 (2019), 115–140. DOI: https://doi.org/10.1016/bs.adcom.2019.06.001

[96] S. Suriadi, R. Andrews, A. H. M. Ter Hofstede, and M. T. Wynn. 2017. Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. *Inf. Syst.* 64 (2017), 132–150. DOI: https://doi.org/10.1016/j.is.2016.07.011

[97] Nick Szabo. 1997. Formalizing and securing relationships on public networks. *First Monday* 2, 9 (Sep. 1997). DOI: https://doi.org/10.5210/fm.v2i9.548

[98] Faizan Tariq and Ricardo Colomo-Palacios. 2019. Use of blockchain smart contracts in software engineering: A systematic mapping. In *Proceedings of the Computational Science and Its Applications (ICCSA'19)*. Springer International Publishing, Cham, 327–337. DOI: https://doi.org/10.1007/978-3-030-24308-1_27

[99] T. Tateishi, S. Yoshihama, N. Sato, and S. Saito. 2019. Automatic smart contract generation using controlled natural language and template. *IBM J. Res. Dev.* 63, 2/3 (Mar. 2019), 6:1–6:12. DOI: https://doi.org/10.1147/JRD.2019.2900643

[100] Paul J. Taylor, Tooska Dargahi, Ali Dehghantanha, Reza M. Parizi, and Kim-Kwang Raymond Choo. 2020. A systematic literature review of blockchain cyber security. *Dig. Commun. Netw.* 6, 2 (2020), 147–156. DOI: https://doi.org/10.1016/j.dcan.2019.01.005

[101] Tezos. 2018. Michelson: The language of smart contracts in I - Semantics. Retrieved October 30, 2020 from https://tezos.com/.

[102] Nachiappan Valliappan, Solène Mirliaz, Elisabet Lobo Vesga, and Alejandro Russo. 2018. Towards adding variety to simplicity. In *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice*, Tiziana Margaria and Bernhard Steffen (Eds.). Springer International Publishing, Cham, 414–431. DOI: https://doi.org/10.1007/978-3-030-03427-6_31

[103] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, and F. Wang. 2018. An overview of smart contract: Architecture, applications, and future trends. In *Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV'18)*. 108–113. DOI: https://doi.org/10.1109/IVS.2018.8500488

[104] Ingo Weber, Xiwei Xu, Régis Riveret, Guido Governatori, Alexander Ponomarev, and Jan Mendling. 2016. Untrusted business process monitoring and execution using blockchain. In *Business Process Management*, Marcello La Rosa, Peter Loos, and Oscar Pastor (Eds.). Springer International, Cham, 329–347. DOI: https://doi.org/10.1007/978-3-319-45348-4_19

[105] T. Weingaertner, R. Rao, J. Ettlin, P. Suter, and P. Dublanc. 2018. Smart contracts using blockly: Representing a purchase agreement using a graphical programming language. In *Proceedings of the 2018 Crypto Valley Conference on Blockchain Technology (CVCBT'18)*. 55–64. DOI: https://doi.org/10.1109/CVCBT.2018.00012

[106] Jeffrey Wilcke. 2015. Mutan Language. Retrieved October 30, 2020 from https://github.com/obscuren/mutan.

[107] Aeternity Workgroups. 2017. Sophia Introduction. Retrieved November 2, 2020 from https://github.com/aeternity/aesophia/blob/lima/docs/sophia.md.

[108] Xtext.org. 2019. Introduction to Xtext. Retrieved November 25, 2020 from https://www.eclipse.org/Xtext/.

[109] Zheng Yang and Hang Lei. 2018. Lolisa: Formal Syntax and Semantics for a Subset of the Solidity Programming Language. arxiv:cs.PL/1803.09885. Retrieved from https://arxiv.org/abs/1803.09885.

[110] Hirai Yoichi. 2017. Morphing Smart Contracts with Bamboo. Retrieved Nov 5, 2020 from https://github.com/jchavarri/bamboo.

[111] Xiao Yue, Huiju Wang, Dawei Jin, Mingqiang Li, and Wei Jiang. 2016. Healthcare data gateways: Found healthcare intelligence on blockchain with novel privacy risk control. *J. Med. Syst.* 40, 10 (2016), 218. DOI: https://doi.org/10.1007/s10916-016-0574-6

Received April 2020; revised December 2020; accepted March 2021

Appendix 2

II

V. Dwivedi, A. Norta, A. Wulf, B. Leiding, S. Saxena, and C. Udokwu. A formal specification smart-contract language for legally binding decentralized autonomous organizations. *IEEE Access*, 9:76069–76082, 2021

Received May 10, 2021, accepted May 15, 2021, date of publication May 19, 2021, date of current version May 28, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3081926

A Formal Specification Smart-Contract Language for Legally Binding Decentralized Autonomous Organizations

VIMAL DWIVEDI[1], ALEX NORTA[1], (Member, IEEE), ALEXANDER WULF[2], BENJAMIN LEIDING[3], SANDEEP SAXENA[4], (Senior Member, IEEE), AND CHIBUZOR UDOKWU[1]

[1]Department of Software Science, Tallinn University of Technology, 12616 Tallinn, Estonia
[2]SRH Berlin University of Applied Sciences, 10587 Berlin, Germany
[3]Institute for Software and Systems Engineering, Clausthal University of Technology, 38678 Clausthal, Germany
[4]Galgotias College of Engineering and Technology, Greater Noida 201310, India

Corresponding author: Vimal Dwivedi (vimal.dwivedi@taltech.ee)

This work was supported by the Erasmus+ Strategic Partnerships Project Blockchain for Entrepreneurs - a Non-Traditional Industry 4.0 Curriculum for Higher Education under Grant 2018-1-RO01-KA203-049510.

ABSTRACT Blockchain- and smart-contract technology enhance the effectiveness and automation of business processes. The rising interest in the development of decentralized autonomous organizations (DAO) shows that blockchain technology has the potential to reform business and society. A DAO is an organization wherein business rules are encoded in smart-contract programs that are executed when specified rules are met. The contractual- and business semantics are sine qua non for drafting a legally-binding smart contract in DAO collaborations. Several smart-contract languages (SCLs) exist, such as SPESC, or Symboleo to specify a legally-binding contract. However, their primary focus is on designing and developing smart contracts with the cooperation of IT- and non-IT users. Therefore, this paper fills a gap in the state of the art by specifying a smart-legal-contract markup language (SLCML) for legal- and business constructs to draft a legally-binding DAO. To achieve the paper objective, we first present a formal SCL ontology to describe the legal- and business semantics of a DAO. Secondly, we translate the SCL ontology into SLCML, for which we present the XML schema definition. We demonstrate and evaluate our SLCML language through the specification of a real life-inspired Sale-of-Goods contract. Finally, the SLCML use-case code is translated into Solidity to demonstrate its feasibility for blockchain platform implementations.

INDEX TERMS Blockchain, smart contract, decentralized autonomous organization, ontology, smart contract language, business process, B2B.

I. INTRODUCTION

Blockchain technologies have spawned new business operations and management models since the former overcome information sharing and resource integration in traditional business management [1]. The latter have relied on a centralization model with hierarchical structures, consequently lacking transparency in inter-organizational processes and trust among participants. Decentralization is an alternative way of conducting business where transactions are distributed and duplicate copies of each transaction are shared with the participants [2]. Blockchain technologies shift the notion of transferring decision-making power and -functions from a single authority to operational units at multiple levels within an organization. Blockchain is a peer-to-peer digital- and distributed ledger where records of business operations are stored in an encrypted manner. Each duplicate record is distributed to every participant's ledger and thus, no trust among the participants is required in a business transaction. Besides, blockchain removes centralized institutions to validate transactions that are managed by a peer-to-peer network [3]. The recent development of blockchain technology empowers and transforms business activities due to the decentralization and disintermediation of power structures. Thus, the immutable traceability of blockchain technology establishes trust among the collaboration participants and reduces cost and time

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Zakarya.

in business transactions by eliminating the need for intermediaries [4].

Information exchange is critical in collaboration between multiple organizations. For example, in a supply chain and X-road [5], numerous collaborative parties are committed from production to delivery, and the integration of processes of each involved party requires widespread information interchange [6]. The lack of consistent information exchange poses a collaboration challenge for inter-organizational business processes [7]. Blockchain technology controls the execution of inter-organizational business processes through smart contracts and enables decentralized autonomous organizations (DAO) [8]. A DAO is an organization, or corporation whose business activities are automated as per agreeing to rules and principles that are specified in programming code [9]. The recent DAOs (such as The DAO) are controlled by the software community, seeking to re-implement traditional decision-making rules through blockchain technology [9]. The DAOs‘ regulations and transactions are stored on a blockchain, which increases the transparency among stakeholders while the execution of the said DAOs‘ rules is controlled by programming code.

A smart contract is a digital agreement in which the participant's rights and obligations are specified in a program-code, including the agreeing rules within which participants carry out these rights and obligations [10]. The concept of a smart contract is first time introduced in seminal work [11] by Nick Szabo in 1997. According to Szabo, "smart contracts can facilitate all steps of contracting processes". Thus, the search, negotiation, performance, adjudication, commitment could be represented in smart contracts. Still, the Szabo vision has surged in the last few years due to the increased availability of IoT devices and the latest evolution of blockchain technology, rendering smart contracts a viable business concept [12]. Blockchain provides an encrypted ledger for smart contracts that are essential for the integrity- and security assurance of smart-contract executions. Ethereum blockchain[1] invented an ethereum virtual machine (EVM) to execute Turing-complete scripts and run decentralized applications. The first implementation of the DAO crowdfunding project, so-called "The DAO," was developed on April 30, 2016, on the Ethereum blockchain to provide business solutions [13]. The idea of implementing "The DAO" was to provide a novel business model where the investor, or shareholder can run both commercial and non-profit enterprises without having a traditional management structure. In the starting phase of its outset, The DAO obtained the notice from media on growing the correspondent of 168 million dollars from various investors to establish the world's largest crowdfunding project. This DAO was maliciously misused by an attacker who stole 50 million dollars due to a flaw in the written DAO smart-contract code. The other core obstacle in the evolution of The DAO is the appropriate legal foundation. Consequently, the concept of The DAO failed due to the application of traditional contract law.

In our work, we consider DAOs to be virtual enterprises (VE), where each enterprise is a collaborating part of a network with peers and is governed by smart contracts that limit the behaviour of each enterprise [14]. Each enterprise is an autonomous, decentralized, and self-organizing network that enables a faster and more cost-effective response to market changes. Enterprises, in the context of DAOs, are peers, or agents that perform the specific functions required in the collaboration lifecycle. Humans and software agents can work together via DAOs, or virtual enterprises [15]. DAOs use peer-to-peer (P2P) computing without any clouds/servers in a loosely coupled collaboration lifecycle in which software agents participate in smart contracting- setup [16], enactment [17], potential rollbacks, and, finally, orderly termination. This lifecycle facilitates the selection of DAO-provided and used services, smart-contract negotiations and behaviour monitoring during enactment with the possibility of breach management [18]. Participants, or parties involved in organizational collaborations are known as human actors who are assigned different roles based on the tasks (functions) they perform in a collaboration [19]. Furthermore, smart objects such as belief-desire-intention (BDI) agents can be combined with smart contracts to collaborate as self-aware DAOs [20].

We discover that several workarounds, for example, SPECS [21], Symboleo [22], SmaCoNat [23], have been published in the scientific literature to develop legally-binding SCLs. The existing research is limited to specify smart legal contracts only for simple business contracts. However, they are not feasible to formulate complex collaborative business contracts (such as DAOs) in a legally-relevant way. Therefore, this paper fills the gap by answering the research question, i.e., how to develop a formal-specification language for the purpose of legally-binding DAO collaboration. The contributions of the paper are first the development of a SCL ontology[2] that comprises concepts and properties for the design of legally relevant DAO collaboration. Secondly, we translate the SCL ontology into the smart legal contract markup language (SLCML) for which we give the schema definition.[3] SLCML allows to define the configuration of a smart contract (instead of its development) for DAO collaboration. To reduce the complexity of the main research question and establish a separation of concerns, we deduce the following sub-research questions. What is the formal semantics to define the legal aspects for a business process? What is the machine-readable language conversion based on the ontology? What is the feasibility-evaluation approach of the language for a use case?

The structure of the paper is as follows. Section II discusses the automobile running case for this paper in which we show the conflict of rights and obligations among the collaborating parties. Further, we explain the preliminaries,

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

[2] shorturl.at/gxFKT

[3] shorturl.at/uBHR6

which prepares the reader to comprehend the subsequent sections. In Section III, we represent the formally-verified common SCL ontology with the objective of specifying each type of legally binding collaborative business smart contracts. We present the syntax and structure of SLCML in Section IV describing the translation of contractual concepts and properties of the SCL ontology into the SLCML schema. Section V defines the feasibility evaluation of SLCML and then show the examples of the SLCML, accompanied by a discussion of proof-of-construction applications. Section VI discuss the solidity code translation from SLCML code. Section VII discuss the related work and finally, Section VIII concludes the paper and discuss the future work.

II. MOTIVATING EXAMPLE AND PRELIMINARIES

We present the running case from the automobile industry for legally binding smart-contract elaboration. We assume that a CarMan produces cars and outsource a significant portion of the supply chain to partnering counterparties that behave as service providers, i.e., sellers. Thus, we present the running case in Section II-A and discuss a conflict scenario of rights and obligations among the collaborating parties. Next, the related background literature is described in Section II-B that prepares the reader for subsequent sections.

A. RUNNING CASE

Blockchain can be applied in the automotive industry, such as electric vehicle charging stations [24], toll systems [25], etc. The significant use-case of blockchain is tracking and monitoring the vehicle parts in the automotive supply-chain [26]. The P2P DAO-collaboration model is shown in Figure 1, where a service consumer's in-house process is a so-called business network model (BNM) [27]. A BNM embodies orchestration that is significant to a business setting and comprises legally binding template contracts, which include service types with clearly defined roles. The setup phase of the DAO-collaboration lifecycle includes BNM selection, populate-module, and negotiate-module for setting up smart-contracting preliminaries [16]. BNM selection is an ecosystem for developing service types that can be used in tandem with BNM in a collaborative platform that includes business processes as a service (BPaaS-HUB) [28] in subsets of the internal in-house process [29]. A BPaaS-HUB provides a rapid exploration of business partners for matching services that focuses on finding collaboration parties, determining their identity and learning about their offerings and reputation. Our previously developed eSourcing Markup Language (eSML) [30] serves as the basis for specifying BNM-specifications.

The populate-module affirms the contained service offers against the BNM's service types as it emerges from the breeding ecosystem. A proto-contract emerges at the end of the populate-phase in the DAO-setup lifecycle [16], for the DAO-participants to begin negotiations. All DAO participants collect a smart-contract replica and can vote on one of three options. DAO participants reach an agreement and create the smart contract for subsequent roll out and enactment; a counter offer from only one DAO-participant causes a business-semantic reversal to the creation of the negotiate-module; and finally, a disagreement from only one DAO-participant results in an absolute termination not only of the contract negotiation but also of the DAO setup. The negotiation of service type, service offer, and service role is divided into two stages, which are depicted in [16]. Phase 1 entails the extraction of proto-contracts, while Phase 2 is used for the consensual establishment of smart contracts. According to [31], agent-based negotiation is rapidly progressing and enables semi- to fully automated negotiation.

The populate-module matches the implanted service offers against the BNM's service types that emerge from the breeding ecosystem. A proto-contract emerges at the end of the populate-phase in the DAO setup lifecycle [16], indicating to the DAO-participants to begin negotiations. All DAO participants are assigned a smart-contract replica and can vote on one of three options. DAO participants reach an agreement and create the smart contract for subsequent roll out and enactment; a counter offer from only one DAO-participant causes a business-semantic reversal to the creation of the negotiate-module; and finally, a disagreement from only one DAO-participant results in an absolute termination not only of the contract negotiation but also of the DAO setup. The negotiation of service types, -offers, and -roles is divided into two stages, which are depicted in [16]. Phase 1 entails the extraction of proto-contracts, while Phase 2 is used for the consensual establishment of smart contracts. According to [31], agent-based negotiation is rapidly progressing and enables semi- to fully automated negotiation.

Service offers are matched with service types from the BNM on the external layer of Figure 1. The dashed monitorability- and conjoinment arcs [33] show how the proposed conceptual business processes are connected to the external layers, and these can be realised from a technical point of view with the lightning network [34]. The decentralized lightning network is suitable for micro-payments, allowing instant, high volume transactions without delegating custody of funds to a third party. In Figure 1, the SupTr, SupST, Shipping are the service providers, i.e., DAO-participants where SupTr produces the tires, SupST makes the steering wheels, and Ship is a shipper that delivers the assembled cars, while the CarMan is a service consumer who assembles the shipped car parts to manufacture a car. The collaboration among these entities creates a DAO for manufacturing and exporting cars. A CarMan organizes an internal business process according to various perspectives such as process-control, information-exchange, workforce management, allocation of means of production, and so on. There is reason to acquire services from service providers that are manifold, e.g., the CarMan cannot manufacture the tires with a similar quality, or at a low price per piece, or the production capacity is not sufficient, or required special know-how is lacking, and so on. The very top and bottom of Figure 1 depict the legacy-technology layers where the processes from the

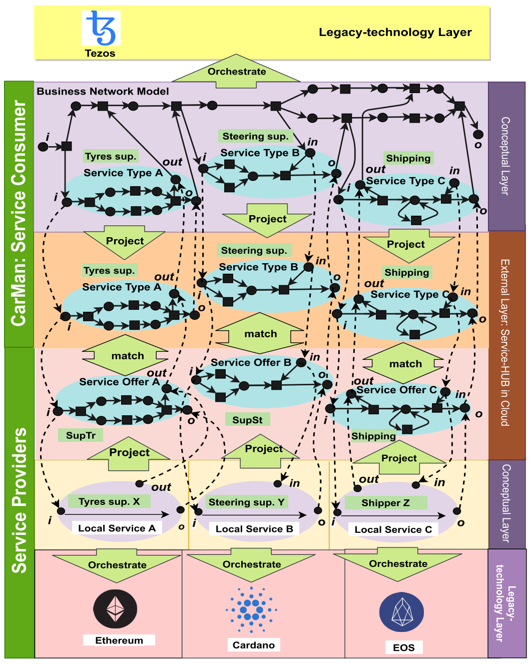


FIGURE 1. DAO-collaborative automotive supply-chain [32].

conceptual layers of the service providers are mapped into smart-contract blockchain systems on the respective internal legacy-technology layer. The tech space of each layer is heterogeneous, although our focus is on the internal legacy layer with the blockchain systems. Thus, on the collaborating parties' respective internal legacy-technology layers, diverse smart-contract blockchain solutions may be used such as Ethereum, Cardano, EOS, Tezos, etc.

We focus only on market exchanges that flow among the entities. CarMan publishes the demand through smart contracts on a blockchain to purchase car parts specifying several criteria such as delivery dates, price, etc. Service providers are notified of CarMan's demand through public blockchain platforms and submit bids, including the state of the car parts. To maintain the confidentiality of bid price and personal data to service providers, a smart contract contains the rules that cannot be altered and opened before the deadline [35]. Furthermore, bids submitted by participants in a public blockchain can be encrypted before being submitted. The key to decrypt the bid is held by a software agent that receives bids. The CarMan chooses then suppliers either manually, or automatically if their specified requirements are met, as details of the collaboration are stored on blockchains. The clauses related to the automotive collaboration are specified in the respective legacy-technology layers to trigger specific events. For example, if the SupTr cannot deliver the car tires to CarMan at a defined time, the SupTr is charged a penalty by smart contracts prior to delivery. In a conventional supply chain, collaborating entities often have less, or no oversight of which entities are accountable for bottlenecks. This oversight is achieved through smart contracts and blockchain technology, where collaborative parties can monitor and track the status of products and transactions. Still, we raise legal- and business challenges that may arise due to the immature SCLs and blockchain technology. For instance, a smart contract releases the funds (ether, bitcoin, etc.) automatically after delivering the car tires to the CarMan and the delivered product does not match the specified requirements of CarMan. The car tires are damaged prior to delivery, and in such a case, CarMan claims compensation, or exchanges the product. The obligation must be imposed to fulfill that compensation on the SupTr. Another case assumes the SupSt sells the steering wheels to CarMan and due to the Shipper's conflict, the product is not delivered within the deadline set by CarMan.

Traditionally, these types of issues can be resolved through the use of a letter of credit in international trade, in which the buyer receives a guarantee that the price of the cargo is not paid unless the seller demonstrates that he fulfills the obligations assigned to him under their underlying sale contract. Furthermore, the seller receives his money, and the bank receives a commission for acting as an intermediary in this transaction [36]. Still, this payment method faces numerous challenges for being a slow and outdated paper-based mechanism that requires both parties to exchange and verify official- and legal documents. Furthermore, this payment method relies solely on the documents to initiate payment, rather than the underlying condition of the goods [37]. The need for 'physical documentation exchanges,' along with the transfer bill of lading and separate correspondence between many different parties, is what renders paper-based letters of credit time-consuming. These can be changed by implementing blockchain, which reduces the time required for credit transactions by allowing an electronic transfer of bills of lading and other requested documents and connecting all parties in a single- and private network, allowing for immediate updates, and eliminating the long lead time for back-and-forth communication among the various parties in letter-of-credit transactions. Still, the properties of contractual semantics in existing smart-contract languages do not exist to draft the blockchain-enabled letter of credit.

B. PRELIMINARIES

In the previous section, we discuss the challenges in writing collaborative smart contracts for the supply chain where parties' rights and obligations must be specified. To formalize the contractual- and business-collaboration concepts and properties, an ontology is a suitable means to conceptualize the knowledge of a particular domain [38], and is used to overcome the conceptual inconsistencies in the blockchain domain [39]. The ontology is a composition of triple sentences, and the latter incorporates purpose, relationship, and object, which allows the practitioner to understand the relationship of concepts in a particular domain. Humans with informatics skills and machines can understand the expressed domain knowledge and information in an ontology. Both can interact with the ontology by explicitly defining the type of concepts, or constraints of its use. We employ the Protégé tool [40] for developing the SCL ontology, which is an open-source ontology editor and comprises a VOWL [41] graphical interface to visualize the relationship among concepts. For checking the inconsistencies of ontologies and identifying the subsumption relationships between classes, the HermiT-tool reasoner is employed [42]. Next, we use the Liquid studio tool for mapping the ontological concepts and properties into XML schema. Liquid Studio[4] includes a comprehensive set of tools for XML and JSON implementations, as well as data mapping and transformation tools (such as XSLT-, XQuery editor, and so on), and a graphical XML schema editor for visualizing, authoring, and navigating complex XML schema. The former offers an effective logical view of the XML schema, allowing for intuitive editing while still allowing you to use all aspects of the W3C XML schema standard.

Next, we discuss the required set of concepts and properties for specifying a legally-relevant and contractual-based collaboration specification language.

III. ONTOLOGICAL CONCEPTS AND PROPERTIES

We develop the SCL ontology comprising the concepts and properties that allow the formulation of smart contracting DAO collaboration in a legally-relevant perspective. We expand the set of concepts and properties for the SCL ontology, considering our prior work about the collaboration-model in [30]. In our previous work, the eSourcing framework is defined to specify and verify harmonized B2B process collaborations [43]. Based on the concept of eSourcing, the eSourcing ontology [30] is designed to configure collaborating parties and their services in a decentralized, contractual collaboration model. Still, the eSourcing ontology lacks legally relevant contractual properties as proposed by the SCL ontology. A contract in the SCL ontology and SLCML includes the legal elements of contractual collaboration, i.e., the rights, obligations, and performances. Rights are fundamental normative regulations for what is permitted, or owed to individuals under the legal system, social convention or ethical theory [44]. Contract obligations are those duties for which each partner in a contract agreement is legally liable. Performance of the Contract means that the parties have fulfilled their respective obligations under the contract. In this paper, we only discuss the legal aspects of the SCL ontology and the rest about the collaboration model, we refer the reader to [30] for further information about the collaboration model.

Since contracts can be of different types, the realm and range of each type differ vastly [45]. It is difficult to express the entire spectrum of contracts in a single ontology because the latter is far too large and diverse to be useful. To capture the full range of business-related contracts within a unified model, a multi-tiered contract ontology that progresses

[4]Liquid Studio | Home

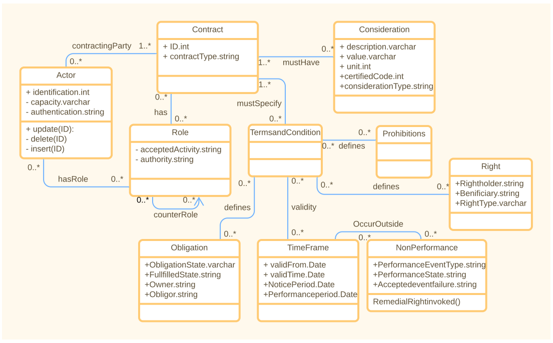


FIGURE 2. Outline for the upper-level smart-contract ontology.

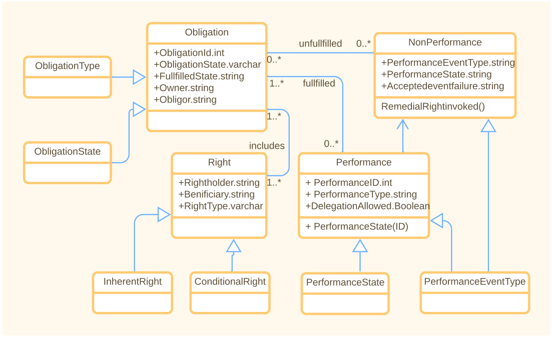


FIGURE 3. Rights and obligations.

from abstract to specific meta data definitions to stratifications is proposed. The two layers of the multi-tier SCL ontology is identified as presented below; other extensions and layers might be possible. The upper core layer depicts the broad configuration of smart contracts applicable over most of the widespread types of contracts. The fundamental concepts such as rights, obligations, and roles are considered building blocks for defining all types of business contracts, as presented in Figures 2 and 3. The specific domain layer is a collection of various contract-type ontologies such as employment contracts, sale of goods, etc. As shown in Figure 4, every contract-type retains every underlying characteristic of the upper-layer and then excels in the particular information specific to the contract domain.

A. UPPER CORE LAYER OF SMART CONTRACTS

We illustrate the upper core layer of legally-relevant smart-contract DAOs depicted in Figures 2 and 3 through the business setting. Assuming a running-case scenario from Section II-A where SupTr and SupST promise to provide the tires and steering wheels respectively to CarMan and on the other hand, the latter promises to return the sum of money. The promise is a declaration of devotion to perform activities or set of actions, such as supplying tires and steering wheels. When the promises are made with legal intent to substantiate in any judiciary, the former becomes a legal obligation. The legal testimonials of the promises (viz. obligations) originate from the contracting parties (viz. actors) are specified in the contracts, comprising details of composing the obligations, admitted limits, and performance measures such as time, venue, etc. Actors are the offeror, offeree, and mediators who perform their roles specified in smart contracts. In our running example, CarMan is an offeror who has the buyer's role, and SupTr, SupST is the offeree who has the service provider's role. CarMan creates an offer to buy the tires and steering wheels to SupTr and SupSt, respectively. A smart contract is legally binding if the contracting parties have the necessary capacity, or competence to enter into the contract [46]. If a party is unable to understand the contract, or is presumed to be unable to do so, the party lacks the competence, or capacity to enter into a contract. A person lacks legal capacity who is insane, or under a certain age, for example, may be considered incompetent to enter into a contract. Several collaborating DAOs, such as SupTr, SupSt, and CarMan, must have legal capacity. A DAOs' legal status was recently established in Wyoming.[5]

A consideration is a benefit that must be negotiated between the parties and is the principal cause for a party to enter into a contract. Considerations can also be as simple as pledging to repair a leaking roof or committing not to do something. Tires and steering wheels, for example, are contract considerations for which CarMan, SupTr, and SupSt have entered into a contract. The delivery of tires and steering wheels, as well as transfer of ownership, constitute a performance of the sales contract. Considerations are also just a commitment to fix a leaky roof, or a pledge not to do something.[6] A consideration equally occurs if CarMan signs a contract with SupTr under which CarMan does not order other brands of tires except Goodyear, and SupTr pays CarMan $500 per year for adhering to this agreement. The promise of the sellers, i.e., the sale of the tires and steering wheels, is an obligation that is fulfilled when the real business activities of supplying the tires and steering wheels are carried out in return for money. CarMan is a beneficiary, or claimant who receives the consideration, or is the individual to whom the business operations are performed. Finally, smart contracts specify the terms and conditions under which the agreed performances are carried out. Typically, contractual performance takes place as stipulated and agreed in the contracts. If the performance is not enforced within the expected timeframe, or executed inadequately, the obligation state becomes unfulfilled. On behalf of the promised party, the occurrence of the non-performance event stimulates certain pre-agreed rights. Assume that the SupTr does not deliver the tires to CarMan under the terms and conditions agreed upon. CarMan seeks a remedy for a penalty, or interest; or may prefer to terminate as per the contract. Alternatively, CarMan may refrain from any punishing actions and resolve the conflict in a calming manner with mutual consensus on how to proceed. The service provider is obligated to fulfill any type of remedy (i.e., reconciliatory promise) as requested by the CarMan. The reconciliatory promise is considered to complete the initial commitment.

We present a simple case study above, where we observe that obligations may trigger further obligations and rights. In the same way, rights may activate new obligations, etc. In the next section, we will discuss the obligation types that are extracted from the upper-layer ontology.

B. SPECIFIC DOMAIN LAYER

The contract statements are informative, declarative, or performative, as discussed in [47], [48]. Informative statements recognise several details, such as the identity of the parties, which law can be enforced, the subject matter of the contract, and so on. Declarative statements express the intention, or condition that changes the state through the performance of the specified conditions. The former are usually of several kinds, such as rights, obligations and prohibitions. Obligations are mandatory statements in contracts that include the obligation owner who is the recipient of the obligation and the obligor, or debtor who performs the obligation. The obligor, or debtor is obliged to execute the obligation condition once and only once in each execution of the contract. Similar to the obligations, rights have right holders and beneficiaries, while the rights are performed by the rights holders. The execution of right is optional and may be performed under specific circumstances depending on the performance of obligations. Prohibitions are statements describing which action should not be taken, or which actions are unacceptable to either party, or both parties.

The obligations are bound to their performative and non-performative events in order to fulfill the former. Based on the nature of the obligations' fulfillment execution, the latter is categorized as primary, reciprocal, conditional, and secondary, as shown in Figure 4. Primary obligations are fulfilled if the primary objectives of the contract are met. For example, the primary obligation of SupTr and CarMan is fulfilled when SupTr delivers the tires in accordance with the contract, or CarMan accepts and pays for tires as ordered. The reciprocal obligation may in itself be the primary obligation, but the former is also the obligation that the counterparty is required to perform in response to the execution of the latter. The CarMan obligation to pay, for example, is relational to the SupTr obligation to deliver, and vice versa. The responsibility to pay for CarMan is also a primary obligation of the former. A conditional obligation does not have to be met in the normal course of events. Most of the remedial rights and obligations fall into this category. For example, if CarMan does not receive the tires and steering wheels within a specified time frame, CarMan may seek compensation for failed delivery. Correspondingly, the service provider is obligated to deliver the good in addition to an extra penalty fee. Finally, a secondary obligation is a sub-part of a primary obligation and may be activated for additional commitment. For example,

[5]DAO | Legal status

[6]Consideration | Legal Definition

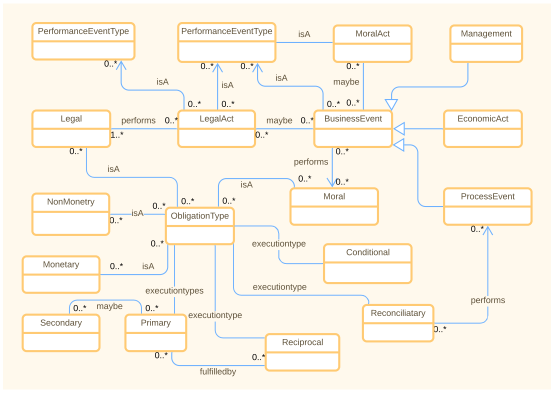


FIGURE 4. Specific domain layer.

SupSt and SupTr are also committed to packaging services that are not legally bound to provide such services.

We also categorize obligations into legal-, business-, and ethical obligations based on the contextual nature of the obligation that requires a particular type of performance. Every declaration in the business contract is legally enforceable and also has legal consequences. Nevertheless, the category of legal obligation is proposed in order to differentiate obligations that require certain specific legal actions to be taken in order to fulfill the latter. Similarly, business obligations are legally binding to categorize those obligations that are specifically related to the performance of the business. Business obligations are classified into monetary and non-monetary obligations. Monetary obligations, e.g., late-payment charges are those dealing with economic, or financial consequences. Furthermore, not all business obligations necessarily have to be financial commitments. Commitments such as CarMan sends orders to buy the steering wheels after contracting, or SupSt is required to arrange for the carrier and notify CarMan, etc., require a business execution. Obligations between CarMan and SupSt, such as tire replacement, logistics carrier arrangement, etc., have no economic implications and we consider these types of obligations to be non-monetary obligations. Legal norms often directly refer to moral- and ethical principles.[7] The contracting parties are thus, legally- and morally obligated to assist their services. For example, service providers are legally- and morally obligated to arrange for the pickup of car components from their premises. Next, we convert the concepts and properties of the SCL ontology into a machine-readable language, i.e., SLCML, for which links are provided in Section I to download the complete ontology and SLCML schema definition.

IV. SLCML: A CONTRACT SPECIFICATION LANGUAGE

The extended SCL ontology comprises the legal concepts and properties of contractual business DAO collaboration. Further, the ontology is verified by the Hermit-reasoner [49], and for the proof-of-concept, the former is translated into a machine-readable language termed SLCML. Our previously developed eSourcing Markup Language (eSML) is implemented on the basis of eSourcing ontology, in which our focus was incorporating a smart-contract collaboration configuration. The development of the eSourcing ontology and eSML answers three key contractual questions, i.e., who-, where-, and what-concepts. Who-concepts identify the contracting parties and where-concepts distinguish the basic aspects of the electronic-contract context, and finally what-concepts define the exchanged values and the related conditions. For further details, we refer to the reader [30]. Still, the legal elements of contracts in SCL are critical for forming a legally binding smart contract. Therefore, we first enhance eSourcing ontology with a law researcher[8] and provide a mature SCL ontology for the advancement of DAO-based smart-contract collaboration. The next step is to map the extended concepts and properties of the SCL ontology into the eSML language for which we use Liquid Studio Tool[9] as an XML schema editor for writing XML documents. The enlarged version of eSML, we call the SLCML. Next, we only discuss the extension part of SLCML, which is not part of the eSML foundation, and provide the link for the reader to download the complete SLCML schema in Section I.

Next, we present the SLCML schema of the upper-level smart contract in Section IV-A. The schema for defining the domain specific contractual properties are presented in Section IV-B.

A. UPPER-LEVEL SMART-CONTRACT DEFINITION

The code extract in Listing 1 defines the legal elements described in the upper layer of legally relevant smart-contract DAOs. The element role in Line 4 defines the role of parties that may be the buyer, the seller, etc., as discussed in Section III. The contractual considerations, along with the variable types, are set out in Line 6 of Listing 1. The value of `minOccurs` and `maxOccurs` in Line 6 shows the amount of consideration required for a legally binding smart contract. In order to specify the terms and conditions in the smart contract, we define the `terms_and_conditions` element in Line 8. The terms and conditions comprise the rights, obligations, prohibitions, and timeframes for which the custom-variable terms and conditions-definition type are defined as shown in Listing 2. Line 8 of the Listing 1 defines the description of the contracting party, followed by the custom type `company_info` which includes the name of the contracting party, the type of legal organization, the company contact information, and so on.

The code extract in Listing 2 is part of the terms and conditions that define the rules and regulations governing the performance of the parties, as discussed in Section III. The rights of the elements are defined in Line 3 along

[7] International chamber of commerce | Home

[8] Alexander Wulf contributed to this paper by supporting the creation of the smart contract law ontology with his legal expertise. He did not contribute towards the written text of the paper.

[9] Liquid Studio | Home

```
<xs:element name="contract">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="role" type="variables_def_section" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="consideration" type="variables_def_section" minOccurs="1" maxOccurs="unbounded"/>
            <xs:element name="terms_and_conditions" type="terms_and_condition_definition" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="party" type="company_info" maxOccurs="unbounded" />
            <xs:element name="mediator" type="company_info" minOccurs="0" maxOccurs="unbounded" />
        </xs:sequence>
            <xs:attribute name="contract_id" type="xs:ID" />
            <xs:attribute name="global_language" type="xs:string" />
            <xs:attribute name="web_service_uri" type="xs:string" />
    </xs:complexType>
</xs:element>
```

LISTING 1. Upper layer of the smart-contract schema.

```
1 <xs:complexType name="terms_and_conditions_definition">
2     <xs:sequence>
3         <xs:element name="right" type="right_type" minOccurs="1" maxOccurs="unbounded" />
4         <xs:element name="prohibitions" type="xs:string" minOccurs="0" />
5         <xs:element name="obligation" type="obligation_category" minOccurs="1" maxOccurs="unbounded" />
6         <xs:element name="time_frame" type="variables_def_section" minOccurs="0" />
7     </xs:sequence>
8 </xs:complexType>
```

LISTING 2. Schema definition of terms and conditions.

with the custom type, i.e., the `right_type` by which the parties can configure the types of right. The `minOccurs` and `maxOccurs` show that parties must choose at least one rights. Terms and conditions may be subject to prohibitions and the definition of prohibitions and are described in Line 4. Line 5 specifies the obligations, along with the `obligation_category`, by which the parties may configure several obligations, as shown in Listing 5. Finally, `time_frame` is defined in Line 6 that shows the expiry of the terms and conditions.

`variables_def_section` is a common variable attribute that contains properties used for all simple- and complex variables in SLCML and defined in Listing 3. The string type is needed to define the string data items. For instance, the role of the contracting party may be defined in string type. The boolean data type is required to support the definition of boolean contract data items. For example, the contract may be legally binding or not, and this is defined by the boolean data type. The integer datatype stores numerical values of contract-id and considerations. Special data types such as money_type and event_type define specific contractual

```
<xs:complexType name="variables_def_section">
        <xs:sequence maxOccurs="unbounded">
            <xs:choice>
                <xs:element name="string_var" type="string_type" />
                <xs:element name="real_var" type="real_type" />
                <xs:element name="integer_var" type="integer_type" />
                <xs:element name="boolean_var" type="boolean_type" />
                <xs:element name="date_var" type="date_type" />
                <xs:element name="time_var" type="time_type" />
                <xs:element name="event_var" type="event_type" />
                <xs:element name="money_var" type="money_type" />
                <xs:element name="external_resource_reference_var" type="external_resource_reference_type" />
                <xs:element name="list_of_events_var" type="list_of_events_type" />
                <xs:element name="list_of_strings_var" type="list_of_strings_type" />
                <xs:any namespace="targetNamespace" />
        </xs:sequence>
    </xs:complexType>
```

LISTING 3. Common variable attributes.

activities. For example, the money_type defines the amount of money from a specific currency, and event_type defines the event that may occur during the contract.

B. OBLIGATION-TYPE DEFINITION

`The obligation_category` consists of the `obligation_type`, `obligation_state`, `performance` and `non-performance` specified in Listing 4. The element `obligation_type` along with custom variable `obligation_type-_definition` is specified in Line 3 by which several obligations are configured. The `obligation_state` is defined in Line 4 to monitor the contract fulfillment process via which an obligation can pass through. In the code example of Listing 4 the definition of `obligation_type_definition` is omitted. The obligation state depends on the performance and non-performance conditions defined in Line 5.

Listing 5 is an example of obligation types from which the parties can configure at least one-, or more obligations. The legal obligation is defined in Line 3 along with the string variable type. Business obligations have monetary and non-monetary implications for which `monetary` and `non-monetary` elements are defined in Lines 4 and 5. Similarly, Line 6 defines the moral obligation along with the string type. We follow a similar approach to define the rest of the obligations, as shown in Listing 5.

V. FEASIBILITY EVALUATION

For our automotive running case, we briefly discuss the SLCML code examples based on the presented SLCML

```
<xs:complexType name="obligation_category">
  <xs:sequence>
    <xs:element name="obligation_type" type="obligation_type_definition" minOccurs="1"/>
    <xs:element name="obligation_state" type="obligation_state_definition" minOccurs="1"/>
    <xs:element name="performance" type="variables_def_section" minOccurs="1" maxOccurs="unbounded"/>
    <xs:element name="non-performance" type="variables_def_section" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

LISTING 4. Schema of obligations category.

```
<xs:complexType name="obligation_type_definition">
      <xs:sequence>
          <xs:element name="legal" type="xs:string" minOccurs="0" />
          <xs:element name="monetary" type="xs:string" minOccurs="0" />
          <xs:element name="non-monetary" type="xs:string" minOccurs="0" />
          <xs:element name="moral" type="xs:string" minOccurs="0" />
          <xs:element name="Primary" type="xs:string" minOccurs="0" />
          <xs:element name="Secondary" type="xs:string" minOccurs="0" />
          <xs:element name="Conditional" type="xs:string" minOccurs="0" />
          <xs:element name="reciprocal" type="xs:string" minOccurs="0" />
          <xs:element name="reconciliatory" type="business_event_types" minOccurs="0" />
      </xs:sequence>
  </xs:complexType>
```

LISTING 5. Schema of the type of obligation.

schema in the previous section. Listing 6 shows an example of defining a legally-binding contract that has a unique ID and cannot be changed throughout the contract enforcement. Line 2 defines the public key of the CarMan wallet and the same hold for the SupSt and SupTr wallet in Line 6 and Line 10, respectively. The name of the parties, i.e., CarMan, SupSt, and SupTr, are defined in Line 3, 7, respectively. CarMan has the service consumer's role described in Line 4. The same applies to SupSt and SupTr, which have the role of the service provider specified in Lines 8 and 12 respectively. Considerations of contracts, such as tires and steering wheels, are presented in Line 14 and 15, for which the parties agree to enter into a contract. Next, terms and conditions include the obligations and rights that are defined in Listing 7 and 8 respectively.

Listing 7 shows an example of the CarMan obligation to renumerate money for tires and steering wheels. The obligation has a name and unique ID that monitors performance, and we consider that to be a monetary obligation. Line 3 enables the obligation state, which means that CarMan receives orders, i.e., tires and steering wheels, and that CarMan has an active obligation to pay money to service providers. SupTr and SupSt are the beneficiaries of the obligations as shown in Line 5 and Line 6 respectively, and CarMan is the obligor

```
1  <contract contract_id="Id1">
2          <party address="03 m6">
3              <name> CarMan </name>
4              <role> Service consumer </role>
5          </party>
6          <party address="32 x7">
7              <name> SupSt </name>
8              <role> Steering wheels provider </role>
9          </party>
10         <party address="31 x7">
11             <name> SupTr </name>
12             <role> Tires provider </role>
13         </party>
14         <consideration> Tires </consideration>
15         <consideration> Steering wheels </consideration>
16         <terms_and_conditions>
17             <obligation/>
18             <right/>
19             <prohibitions/>
20         </terms_and_conditions>
21 </contract>
```

LISTING 6. Contract instantiation for the automotive running case.

who is obliged to perform this obligation as set out in Line 7. We assume that no third party, or mediators are involved in this obligation. The to-do obligation has legal implications for which the CarMan has to act by actually paying the money. The preconditions for the obligations are set out in Line 13 and Line 14, for which CarMan and service providers sign contracts (Act1) and (Act2) and CarMan receives tires and steering wheels. The performance type is the payment that needs to be transferred from CarMan to SupSt and SupSt wallet addresses. Besides, the performance object is defined as the buy with the qualifiers, which is paid for a specific amount within the deadline. The `rule_ conditions` specify the time limit for payment and the purchase-payment plan are set out in Line 18. Finally, a reference is added to the obligation in which a remedy for late payment exists. If CarMan fails to pay the money within the time limit then CarMan has to transfer a defined monetary amount to SupTr.

The code extract of Listing 8 comprises intersecting provisions with the obligation. The rights and obligations are intertwined, which means that if one party asserts its rights, the other party is required to comply. Similar to the obligation in Listing 7, the rights have a beneficiary who can be benefited from the right and an obligor who can enable the right. For example, CarMan receives the defective tires, and in that case, CarMan is the owner of the right to claim the replacement of damaged tires. Consequently, the SupTr is obliged to replace the latter.

Again, we assume that the rights have a name and ID as defined in Line 1. As the service providers have a right to waive the right, for example, the SupTr can convince the CarMan the parts were defective during logistics without his fault. The rights can be changed during the execution of the contract and the compensation is set to false if the SupTr agrees to replace the tires. The state of right is available for direct enactment, and the parties are defined in the same way as in Listing 7. The right-type is set to

```
<obligation_rule tag_name ="paying_invoices"
    rule_id ="0001" changeable ="false" monetary =
    "true">
    <state> enabled </state>
   <parties>
      <beneficiary> SupTr (31 x7 ) </beneficiary>
      <beneficiary > SupSt (31 x7 ) </beneficiary>
      <obligor> CarMan (03 m6 ) </obligor>
      <third_party> nil </third_party>
   </parties>
   <obligation_type>
      <legal_obligation> to-do </legal_obligation>
   </obligation_type>
   <precondition>
      act1 (signed)& Tires (transferred)
   </precondition>
   <precondition>
      act2 (signed) & Steering wheels (transferred
    )
   </precondition>
   <performance_type>
      payment (03 m6,31 x7, buy)
   </performance_type>
   <performance_type>
      payment (03 m6,32 x7, buy)
   </performance_type>
   <performance_object>
      invoice ( buy, amount)
   <performance_object>
   <rule_conditions>
      date ( before delivery of tires)
   </rule_conditions>
   <remedy>
      late_payment_interest (amount,03 m6 ,31 x7)
   </remedy>
   </obligation_rule>
```

LISTING 7. Obligation example for paying car parts.

```
<right_rule tag_name ="Car's_component_replacement
    " rule_id ="0002" changeable ="true" monetary
    ="false">
    <state> enabled </state>
    <parties>
       <beneficiary>
           CarMan (31 x7)
       </beneficiary>
       <obligor>
           SupTr (03 m6)
       </obligor>
       <third_party>
           nil
       </third_party>
    </parties >
    <right_type>
       <conditional_right>
           claim
       </conditional_right>
    </right_type>
    <precondition>
           act1 (signed)& car's components (
    transferred)
    </precondition>
    <performance_type>
           replace (defective car's component)
    </performance_type>
    <action_object>
           car's component ( brand, type,
    serial_number)
    </action_object>
    <rule_conditions>
           deadline (date)
    </rule_conditions>
    <remedy>
           late_replacement_interest (amount, 31
    x7)
    </remedy>
</right_rule>
```

LISTING 8. Right example for replacing a broken car's component.

conditional-right, and the CarMan uses that right as a claim for the replacement of the tires. The right's precondition is to have the contract signed and the parts delivered to the CarMan. The performance type is set to replace the tires described as a performance object with a brand, type, and serial number. After enabling this right, the corresponding obligation on the SupTr must be fulfilled under the specified deadline; otherwise, the CarMan claims the remedy payment of a specific amount.

To date, several online dispute resolutions such as online arbitration, crowd-sourced dispute resolution, and AI-powered resolutions have been proposed in the event that parties do not resolve their disputes themselves [50]. Blockchain communities developed arbitration systems to resolve disputes quickly and efficiently in line with appropriate norms and recognized equitable principles. Sagewise's technology,[10] for example, is incorporated into a smart contract through a coded provision in which consumers pre-set specific parameters, including when and how long the smart contract execution should be delayed, and who resolves any disputes that may arise. As a result, if a dispute arises, this clause allows a party to halt contract execution and activate the Sagewise dispute resolution mode. After that, the party can select from a variety of dispute resolution processes for resolving smart-contract issues and enforcing online judgments.

In the following Section, we will demonstrate how to translate SLCML code into Solidity.

VI. SLCML TO SOLIDITY-CODE TRANSLATION

Our starting point is the SLCML code corresponding to our running case generated in Listing 6, 7, 8. The Solidity use case code in Section VI was not generated by the tool, but it is anticipated that it will be generated once the tool is implemented. The transformation rules can be used to translate SLCML code to a choreography model, which is then translated to Solidity code using a Caterpillar [51]. Caterpillar is a fully accessible Blockchain-based BPM system which converts BPMN-modelled business processes into Solidity-based smart contracts. Still, we do not discuss the transformation rules because they are beyond the scope of the paper. We only discuss the Solidity code presented in Listing 9 that contains an excerpt from the generated smart contract. To begin the task execution with rights and obligations, our smart contract, ‘‘Automotive_SupplyChain’’ contains four events and four solidity functions. Lines 3 to 8 of Listing 9 represent global variables and data pertaining to the process state is stored on-chain. As defined in lines 9 to 17, the list of SupSt,

[10]SagewiselDispute resolution

```
1  pragma solidity ^0.4.16;
2  contract Automotive_SupplyChain{
3      uint public role;
4      uint funds;
5      uint tires_quantity;
6      uint steering_wheels_quantity;
7      uint public consideration;
8     .....
9       struct CarMan{address Carman; uint role; }
10      struct SupSt {address SupST; uint role;  }
11      struct SupTr{address SupST;uint role;  }
12 event tires_Supply (uint _quanity, address CarMan,
        address SupTr);
13 event steering-wheels_Supply (uint _quanity,
       address CarMan, address SupST);
14 event notifyObligationBreach (*Define Type*
       obligaton, address contract );
15 function notify(*Define Type* obligaton, address
       CarMan){
16     //TODO: Implement code to notify obligation
       breach for target contract address
17     notifyObligationBreach(obligation type,
       contract);}
18  function release(uint Tire_quanity, address
       CarMan ){
19      // TOD: Implement code to release tires to
       the CarMan. }
20      function release(uint steering_wheel_quanity,
        address CarMan ){
21      // TOD: Implement code to release steering
       wheels to the CarMan. }
22  event claimParcel(*Define type* right, address
       contract);
23  function replace_parcel(*Define type* right,
       address contract){
24      //TODO: Implement code to activate right for
       target contract address }
25     modifier precondition(){
26          //Check the condition
27          uint beneficiary;
28          uint obligor;
29          if(!paybeforedeadline){
30              release(tier_quantity, CarMan);
31              producer.send(funds); }
32          else
33          {  _;  } } }
34  // TODO: check precondition for steering wheels.
```

LISTING 9. Automotive supply chain.

CarMan and SupTr variables is declared in struct, which can be accessed with a single pointer name throughout the contract. In Line 18, a further event for performance type, i.e., tires supply, is implemented, containing parameters such as tires quantity, CarMan address, and SupTr address to which track the delivery of tires and steering wheels. Similarly, an event for performance type, i.e., steering wheels, is implemented in Line 19, along with the wheel quality, CarMan address, and SupSt address, which track the delivery of wheels. Lines 20 to 23 implement the notifyObligation-Breach event and associated function for tracing SupTr and SupSt obligations. Similarly, an event for rights is introduced in Lines 28-30 in the event that a party seeks compensation. Following that, a modifier precondition is used to release the product if the payment is received before the deadline.

VII. RELATED WORK

Existing SCLs such as Solidity, Serpent and so on are developed from an IT perspective where the programmer writes a machine-readable code without the knowledge of the contract domain. Still, we observe that existing research focuses on the development of SCLs to specify legally binding smart contracts. In [21], researchers propose a specification language (SPESC) to define the configuration of a smart contract (rather than its implementation) for the purpose of collaborative design. In SPESC, smart contracts are considered to be a combination of IT experts, domain practitioners and business, or financial transactions. Using SPESC, real-world contract utilities, such as the role of the party, the set of terms and conditions, etc., can be specified in smart contracts. Nevertheless, SPESC does not address many aspects of contracts, such as obligation states, categories of rights and obligations, etc., but instead focuses on modelling legal relations (legal positions). In [22], researchers propose a formal specification language (Symboleo) reflecting obligations and powers, using domain concepts and axioms. Symboleo specifications include rights and obligations that can be monitored on a run-time basis. In addition, formal semantics is introduced to describe the life-cycle of contracts, obligations and authorities on the basis of state charts. Symboleo is sufficiently expressive to represent many types of real-life contracts, but Symboleo does not express the concepts and properties of collaborative contracts.

In [52], the researcher addresses the challenges of formalizing contracts written in natural languages in machine-readable languages. In addition, the contract modeling language (CML) is proposed for modelling and specifying unstructured legal contracts covering a wide range of common contract situations. CML specifies a natural-language comparable clause grammar that resembles real-world contracts, but this research does not address transaction rules and is not sufficient to formalise any type of contracts (viz. domain completeness).

In [23], researchers argue that human contract intentions are mostly defined in natural-language, which is easy to understand but highly ambiguous and subject to interpretation. In addition, a methodology is proposed to develop a high-level specification that achieves common understanding through natural-language phrases and is compiled directly into machine instructions. Still, this research focuses mainly

TABLE 1. Evaluation our specification language against existing SCLs.

SCL	CC*	LS*	DC*	TR*
SLCML	+	+	+	+
SPESCS	-	+	-	+
ADICO	-	+	-	+
Symboleo	-	+	+	+
CML	-	+	-	+
SmaCoNat	-	+	-	+
iContractML	-	-	+	+
ContracT	-	+	+	-
BPSL	+	-	+	-

* *CA* [Contractual collaboration] | *LS* [Legal semantics] | *DC* [Domain completeness]
* *TR* [Transaction rules]

on the readability and safety of smart contracts and does not express the collaborative contractual suitability and completeness of the domain. In [53], researchers find it difficult to implement smart contracts due to the complexity and heterogeneity of the underlying platforms. In addition, the blockchain-independent smart-contract modelling language (called iContractML) is proposed to relieve developers' stress from addressing the particular complexity of blockchains. CML enables blockchain developers to concentrate on the business process instead of the syntactical specifics of each blockchain platform. The focus and scope of CML is completely different from our research. Attributes Deontic AIm Conditions (ADICO) [54] is a DSL developed in Scala that converts domain-specific constructs of smart contracts to simpler concepts. In [55], a contracT tool has been developed that annotates the legal-contract text using a legal-contract ontology. Still, the proposed ontology is not mature enough to develop collaborative smart contracts. This study [56] develops a framework for dynamic binding of parties to collaborative process roles and an appropriate language for binding policy specifications. The proposed language is equipped with Petri-net semantics, which enables the verification of policy consistency.

The above information is described in Table 1 and the essential aspect is shown when thinking about developing SCLML. To evaluate the SCLs, we score them with '+' or '−' operators for each of the four parameters. The former indicates that the SCL has a specific property, whereas the latter indicates that the property does not exist in the corresponding SCL. The result of the table shows that a lot of research is being done in the area of legal smart-contract specification. Still, we address the gap that the solutions address in immaturely, revealing that existing methodologies are limited to the design of all types of real-world contracts. For example, the prior research is not sufficient to specify collaborative- and legally binding smart contracts.

VIII. CONCLUSION

This paper presents the ontological concepts and properties that are critical for developing legally-binding DAOs. We extend our previous work in which the specification of DAO collaboration is discussed and only show the legal element for this paper, which is essential for specifying legal DAOs. An ontology is developed in the OWL language and verified through the HermiT reasoner. The ontology is an input for the development of the SLCML. We map the extended concepts and properties of the SCL ontology into the eSML language. The enlarged version of eSML, we call the SLCML. For this paper, we only discuss the extension part of SLCML, which is not part of the eSML foundation. We provide a code example based on an automotive case study that ensures the language comprises collaborative legal concepts on the basis of semantic clarity.

We discover that the multi-tiered SCL ontology captures the full range of legally binding business-related contracts in a unified model. The upper-core layer depicts the broad configuration of smart contracts applicable to most widespread types of contracts. The specific domain layer is the collection of different types of contracts, such as employment contracts, sale of goods, etc. Blockchain-based smart-contract technology could be used to address the core issues that arise in the context of temporary employment [57], in order to safeguard employees and prevent competition from being distorted in favor of corporations that aim to exploit illegal workers. Each type of contract inherits all the core functions of the upper layer and then specializes in the particular knowledge specific to the contract domain. SLCML adopts a real-life contracting foundation where collaborating parties use their legal properties in decentralized collaborations. SCLML is implemented based on our previously developed eSourcing Markup Language (eSML) in which our focus is incorporating a smart-contract collaboration configuration.

As future work, we aim that the contract ontology can be further developed to achieve domain completeness. In addition, we plan to develop a tool-supported process to transform SLCML contract specification into smart-contract code, e.g., Solidity, and to carry out more case studies with SLCML in blockchain research projects. A formal analysis approach to the specification of SLCML could be developed; we plan to build a translator for the automatic conversion of SLCML instantiations into a larger set of blockchain-based language.

REFERENCES

[1] X. Pan, X. Pan, M. Song, B. Ai, and Y. Ming, "Blockchain technology and enterprise operational capabilities: An empirical test," *Int. J. Inf. Manage.*, vol. 52, Jun. 2020, Art. no. 101946.

[2] P. Vigna and M. J. Casey, *The Age of Cryptocurrency: How Bitcoin and Digital Money are Challenging the Global Economic Order*. New York, NY, USA: St. Martin's Press, 2015.

[3] C. Catalini and J. S. Gans, "Some simple economics of the blockchain," *Commun. ACM*, vol. 63, no. 7, pp. 80–90, Jun. 2020.

[4] Y. Chen and C. Bellavitis, "Blockchain disruption and decentralized finance: The rise of decentralized business models," *J. Bus. Venturing Insights*, vol. 13, Jun. 2020, Art. no. e00151.

[5] R. Saputro, I. Pappel, H. Vainsalu, S. Lips, and D. Draheim, "Prerequisites for the adoption of the X—Road interoperability and data exchange framework: A comparative study," in *Proc. 7th Int. Conf. eDemocracy eGovernment (ICEDEG)*, 2020, pp. 216–222, doi: 10.1109/ICEDEG48599.2020.9096704.

[6] C. Di Ciccio, A. Cecconi, M. Dumas, L. García-Bañuelos, O. López-Pintado, Q. Lu, J. Mendling, A. Ponomarev, A. B. Tran, and I. Weber, "Blockchain support for collaborative business processes," *Informatik Spektrum*, vol. 42, no. 3, pp. 182–190, May 2019.

[7] R. Eshuis, A. Norta, and R. Roulaux, "Evolving process views," *Inf. Softw. Technol.*, vol. 80, pp. 20–35, Dec. 2016.

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

[9] M. Singh and S. Kim, "Blockchain technology for decentralized autonomous organizations," in *Role of Blockchain Technology in IoT Applications* (Advances in Computers), vol. 115, S. Kim, G. C. Deka, and P. Zhang, Eds. Amsterdam, The Netherlands: Elsevier, 2019, ch. 4, pp. 115–140.

[10] N. Diallo, W. Shi, L. Xu, Z. Gao, L. Chen, Y. Lu, N. Shah, L. Carranco, T.-C. Le, A. B. Surez, and G. Turner, "EGov-DAO: A better government using blockchain based decentralized autonomous organization," in *Proc. Int. Conf. eDemocracy eGovernment (ICEDEG)*, Apr. 2018, pp. 166–171.

[11] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, pp. 1–11, Sep. 1997.

[12] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, and A. Ponomarev, "Caterpillar: A business process execution engine on the ethereum blockchain," *Softw., Pract. Exp.*, vol. 49, no. 7, pp. 1162–1193, May 2019.

[13] C. Jentzsch, "Decentralized autonomous organization to automate governance," White Paper 1.0, Berlin, Germany, Nov. 2016.

[14] N. C. Narendra, A. Norta, M. Mahunnah, L. Ma, and F. M. Maggi, "Sound conflict management and resolution for virtual-enterprise collaborations," *Service Oriented Comput. Appl.*, vol. 10, no. 3, pp. 233–251, Sep. 2016.

[15] L. Sterling and K. Taveter, *The Art of Agent-Oriented Modeling*. Cambridge, MA, USA: MIT Press, 2009.

[16] A. Norta, "Creation of smart-contracting collaborations for decentralized autonomous organizations," in *Perspectives in Business Informatics Research*, R. Matulevičius and M. Dumas, Eds. Cham, Switzerland: Springer, 2015, pp. 3–17.

[17] A. Norta, "Establishing distributed governance infrastructures for enacting cross-organization collaborations," in *Service-Oriented Computing—ICSOC 2015 Workshops*, A. Norta, W. Gaaloul, G. R. Gangadharan, and H. K. Dam, Eds. Berlin, Germany: Springer, 2016, pp. 24–35.

[18] A. Norta, A. B. Othman, and K. Taveter, "Conflict-resolution lifecycles for governed decentralized autonomous organization collaboration," in *Proc. 2nd Int. Conf. Electron. Governance Open Soc., Challenges Eurasia (EGOSE)*. New York, NY, USA: Association for Computing Machinery, Nov. 2015, pp. 244–257.

[19] C. Udokwu and A. Norta, "Deriving and formalizing requirements of decentralized applications for inter-organizational collaborations on blockchain," *Arabian J. Sci. Eng.*, vol. 46, pp. 1–18, Mar. 2021.

[20] A. Norta, "Self-aware smart contracts with legal relevance," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2018, pp. 1–8.

[21] X. He, B. Qin, Y. Zhu, X. Chen, and Y. Liu, "SPESC: A specification language for smart contracts," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 1, Jul. 2018, pp. 132–137.

[22] S. Sharifi, A. Parvizimosaed, D. Amyot, L. Logrippo, and J. Mylopoulos, "Symboleo: Towards a specification language for legal contracts," in *Proc. IEEE 28th Int. Requirements Eng. Conf. (RE)*, Aug. 2020, pp. 364–369.

[23] E. Regnath and S. Steinhorst, "SmaCoNat: Smart contracts in natural language," in *Proc. Forum Specification Design Lang. (FDL)*, Sep. 2018, pp. 5–16.

[24] F. Knirsch, A. Unterweger, and D. Engel, "Privacy-preserving blockchain-based electric vehicle charging with dynamic tariff decisions," *Comput. Sci., Res. Develop.*, vol. 33, nos. 1–2, pp. 71–79, Sep. 2017.

[25] B. Xiao, X. Fan, S. Gao, and W. Cai, "Edgetoll: A blockchain-based toll collection system for public sharing of heterogeneous edges," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr./May 2019, pp. 1–6.

[26] S. E. Chang, Y.-C. Chen, and M.-F. Lu, "Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process," *Technol. Forecasting Social Change*, vol. 144, pp. 1–11, Jul. 2019.

[27] T. Ruokolainen, S. Ruohomaa, and L. Kutvonen, "Solving service ecosystem governance," in *Proc. IEEE 15th Int. Enterprise Distrib. Object Comput. Conf. Workshops*, Aug. 2011, pp. 18–25.

[28] A. Norta and L. Kutvonen, "A cloud hub for brokering business processes as a service: A 'rendezvous' platform that supports semi-automated background checked partner discovery for cross-enterprise collaboration," in *Proc. Annu. SRII Global Conf.*, 2012, pp. 293–302.

[29] R. Eshuis, A. Norta, O. Kopp, and E. Pitkänen, "Service outsourcing with process views," *IEEE Trans. Services Comput.*, vol. 8, no. 1, pp. 136–154, Jan. 2015.

[30] A. Norta, L. Ma, Y. Duan, A. Rull, M. Kõlvart, and K. Taveter, "eContractual choreography-language properties towards cross-organizational business collaboration," *J. Internet Services Appl.*, vol. 6, no. 1, Apr. 2015, Art. no. 8.

[31] R. Lin and S. Kraus, "Can automated agents proficiently negotiate with humans?" *Commun. ACM*, vol. 53, no. 1, pp. 78–88, Jan. 2010.

[32] N. C. Narendra, A. Norta, M. Mahunnah, L. Ma, and F. M. Maggi, "Sound conflict management and resolution for virtual-enterprise collaborations," *Service Oriented Comput. Appl.*, vol. 10, no. 3, pp. 233–251, Oct. 2015.

[33] A. Norta and P. Grefen, "Discovering patterns for inter-organizational business process collaboration," *Int. J. Cooperat. Inf. Syst.*, vol. 16, no. 03n04, pp. 507–544, Sep. 2007.

[34] J.-H. Lin, K. Primicerio, T. Squartini, C. Decker, and C. J. Tessone, "Lightning network: A second path towards centralisation of the bitcoin economy," *New J. Phys.*, vol. 22, no. 8, Aug. 2020, Art. no. 083022.

[35] Y.-H. Chen, S.-H. Chen, and I.-C. Lin, "Blockchain based smart contract for bidding system," in *Proc. IEEE Int. Conf. Appl. Syst. Invention (ICASI)*, Apr. 2018, pp. 208–211.

[36] X. H. Li, "Blockchain-based cross-border E-business payment model," in *Proc. 2nd Int. Conf. E-Commerce Internet Technol. (ECIT)*, Mar. 2021, pp. 67–73.

[37] E. M. Al-Amaren, C. Ismail, and M. Nor, "The blockchain revolution: A gamechanging in letter of credit (L/C)," *Int. J. Adv. Sci. Technol.*, vol. 29, no. 3, pp. 6052–6058, 2020.

[38] A. Maedche and S. Staab, "Ontology learning for the semantic Web," *IEEE Intell. Syst.*, vol. 16, no. 2, pp. 72–79, Mar. 2001.

[39] J. de Kruijff and H. Weigand, "Understanding the blockchain using enterprise ontology," in *Advanced Information Systems Engineering*, E. Dubois and K. Pohl, Eds. Cham, Switzerland: Springer, 2017, pp. 29–43.

[40] M. A. Musen, "The protégé project: A look back and a look forward," *AI Matters*, vol. 1, no. 4, pp. 4–12, Jun. 2015.

[41] S. Lohmann, S. Negru, and D. Bold, "The protégéVOWL plugin: Ontology visualization for everyone," in *The Semantic Web: ESWC 2014 Satellite Events*, V. Presutti, E. Blomqvist, R. Troncy, H. Sack, I. Papadakis, and A. Tordai, Eds. Cham, Switzerland: Springer, 2014, pp. 395–400.

[42] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "HermiT: An OWL 2 reasoner," *J. Automated Reasoning*, vol. 53, no. 3, pp. 245–269, Oct. 2014.

[43] A. Norta and R. Eshuis, "Specification and verification of harmonized business-process collaborations," *Inf. Syst. Frontiers*, vol. 12, no. 4, pp. 457–479, Apr. 2009.

[44] S. Vogenauer and J. Kleinheisterkamp, Eds., *Commentary on the UNIDROIT Principles of International Commercial Contracts (PICC)*. Oxford, U.K.: Oxford Univ. Press, 2009.

[45] A. J. Wulf, "Institutional competition of optional codes in European contract law," *Eur. J. Law Econ.*, vol. 38, no. 1, pp. 139–162, 2014.

[46] G. Governatori, F. Idelberger, Z. Milosevic, R. Riveret, G. Sartor, and X. Xu, "On legal contracts, imperative and declarative smart contracts, and blockchain systems," *Artif. Intell. Law*, vol. 26, no. 4, pp. 377–409, Dec. 2018.

[47] R. M. Lee and S. D. Dewitz, "Facilitating international contracting: AI extensions to EDI," *Int. Inf. Syst.*, vol. 1, no. 1, pp. 94–123, 1992.

[48] Y.-H. Tan and W. Thoen, "Modeling directed obligations and permissions in trade contracts," in *Proc. 31st Hawaii Int. Conf. Syst. Sci.*, vol. 5, 1998, pp. 166–175.

[49] I. Horrocks, B. Motik, and Z. Wang, "The hermit owl reasoner," in *Proc. CEUR Workshop*, I. Horrocks, M. Yatskevich, and E. Jiménez-Ruiz, Eds, 2012, vol. 858, no. 3, pp. 245–269.

[50] A. Schmitz and C. Rule, "Online dispute resolution for smart contracts," *J. Dispute Resolution*, vol. 2019, no. 2, pp. 103–126, 2019.

[51] O. López-Pintado. (2021). Orlenyslp/Caterpillar. GitHub. [Online]. Available: https://github.com/orlenyslp/Caterpillar

[52] M. Wöhrer and U. Zdun, "Domain specific language for smart contract development," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency*, May 2020, pp. 1–9.

[53] M. Hamdaqa, L. A. P. Metz, and I. Qasse, "IContractML: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms," in *Proc. 12th Syst. Anal. Modeling Conf. (SAM)* New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 34–43.

[54] C. K. Frantz and M. Nowostawski, "From institutions to code: Towards automated generation of smart contracts," in *Proc. IEEE 1st Int. Workshops Found. Appl. Self Syst. (FAS W)*, Sep. 2016, pp. 210–215.

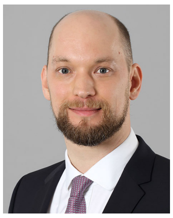
[55] M. Soavi, N. Zeni, J. Mylopoulos, and L. Mich, "ContracT–from legal contracts to formal specifications: Preliminary results," in *The Practice of Enterprise Modeling*, J. Grabis and D. Bork, Eds. Cham, Switzerland: Springer, 2020, pp. 124–137.

[56] O. López-Pintado, M. Dumas, L. García-Bañuelos, and I. Weber, "Dynamic role binding in blockchain-based collaborative business processes," in *Advanced Information Systems Engineering*, P. Giorgini and B. Weber, Eds. Cham, Switzerland: Springer, 2019, pp. 399–414.

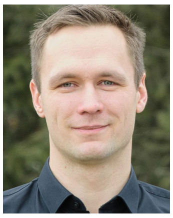
[57] A. Pinna and S. Ibba, "A blockchain-based decentralized system for proper handling of temporary employment contracts," in *Intelligent Computing* (Advances in Intelligent Systems and Computing), vol. 857, K. Arai, S. Kapoor, and R. Bhatia, Eds. Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-030-01177-2_88.

VIMAL DWIVEDI received the master's degree in information technology from Indraprashta University, Delhi. He is currently pursuing the Ph.D. degree with the Blockchain Technology Group, Tallinn University of Technology. He has worked as an Assistant Professor of information technology in India. He is also an Early Stage Researcher with the Blockchain Technology Group, Tallinn University of Technology. He has (co)authored four publications in conference proceedings. His research interests include semantics and ontology development, and legally-relevant smart contract languages development for blockchains.

ALEX NORTA (Member, IEEE) received the M.Sc. degree from the Johannes Kepler University of Linz, Austria, in 2001, and the Ph.D. degree from the Eindhoven University of Technology, The Netherlands, in 2007. He was a Researcher with the Oulu University Secure-Programming Group (OUSPG). He was a Postdoctoral Researcher with the University of Helsinki, Finland. He is currently a Principal Investigator with the Blockchain Technology Group. He is also a Research Member with the Faculty of Software Science, TalTech, Tallinn, Estonia. For the blockchain-tech startups Qtum.org, their respective whitepapers and also serves as an advisor for several other blockchain-tech startups such as Cashaa. His research interests include business-process collaboration, smart contracts, blockchain technology, e-business transactions, service-oriented computing, software architectures, software engineering, ontologies, security, multi-agent systems, distributed business-intelligence mining, e-learning, Agile software engineering, production automation, enterprise architectures, and e-governance. His Ph.D. thesis was partly financed by the IST project CrossWork, in which he focused on developing the eSourcing concept for dynamic inter-organizational business process collaboration.



ALEXANDER WULF is currently a Professor of business law with the SRH Berlin University of Applied Sciences. His research interests include application of empirical methodology to the study of law, the interdependence of law and economics, and the relevance of law and legal institutions for the behaviour of businesses. His research focuses particularly on the empirical analysis of European Union commercial law.



BENJAMIN LEIDING was born in Rostock, Germany. He received the B.Sc. degree in computer science from the University of Rostock, Germany, in 2015, and the M.Sc. degree in Internet technologies and information systems and the Ph.D. degree in computer science from the University of Goettingen, Germany, in 2017 and 2020, respectively. He is currently a Postdoctoral Research Fellow with the Clausthal University of Technology. His research interests include the machine-to-everything economy (M2X Economy), circular economy, distributed systems, and digital identities.

SANDEEP SAXENA (Senior Member, IEEE) received the B.Tech. degree in CSE from UPTU Lucknow, the M.S. degree in information security from IIIT Allahabad, and the Ph.D. degree from NIT Durgapur, West Bengal. He is currently working as an Associate Professor with the Galgotias College of Engineering and Technology, Greater Noida. He have more than 12 years of teaching experience. He had performed the role of a key member in six international conferences as an organizing secretary/organizing chair/session chair. He had written three technical books for UP Technical University, Lucknow. He has published multiple research articles in reputed international journals and conferences. He had published eight international conferences and two SCIE, nine patent published, two Scopus, and six other published in international journals. He is also participating in multiple professional societies, such as IAASSE (Senior Member), CSI, and CRSI.

CHIBUZOR UDOKWU received the master's degree in software science from TalTech, where he is currently pursuing the External Ph.D. degree with the Blockchain Technology Group. He has consulted and helped in designing several blockchain applications for different startups. He has published and coauthored several scientific articles in the blockchain space. His research interests include semantics and ontology development, design and development of blockchain systems, blockchain use-cases, and applications in organizations.

Appendix 3

III

V. Dwivedi and A. Norta. A legal-relationship establishment in smart contracts: Ontological semantics for programming-language development. In M. Singh, V. Tyagi, P. K. Gupta, J. Flusser, T. Ören, and V. R. Sonawane, editors, *Advances in Computing and Data Sciences*, pages 660–676, Cham, 2021. Springer International Publishing

A Legal-Relationship Establishment in Smart Contracts: Ontological Semantics for Programming-Language Development

Vimal Dwivedi(✉) and Alex Norta

Tallinn University of Technology, Akadeemia tee 15 a, Tallinn, Estonia
vimal.dwivedi@taltech.ee, alex.norta.phd@ieee.org

Abstract. Machine-readable smart contracts (SC) on blockchains promise drastic enhancements in collaboration efficiency and effectiveness in that cost- and time reductions can be achieved while the quality of services increases. We address existing shortcomings of SCs that are in tendency incomplete for legal recognition especially to smart-contract-enabled funding rounds, not collaborative business-process reflective and are also not aware of their own processing state to justify the claim of smartness. When conflicts occur, tracing the past performance of conventional contract (CC) execution is very slow and expensive while in addition, CCs are challenging to enforce. On the one hand, the legal status of SCs based funding rounds is currently not clarified and the question arises if SCs comprise the necessary legal- concepts and properties. Current SC solutions do not suffice in those regards. To fill this gap, we develop the smart-legal-contract (SCL) ontology to define the legal- and collaborative business concepts and properties in the SCs. Formal methods, such as Colored Petri Nets (CPNs), are suitable to design, develop and analyze processing state of SCs in order to trace the performance of contractual-rights and obligations. In this work, SCL ontology is formalized using Colored Petri Nets resulting in a verifiable CPN model. Furthermore, we conduct a state-space analysis on the resulting CPN model and derive specific model properties. A running case from the automotive supply chain domain demonstrates the utility and validity of our approach.

Keywords: Smart contract · Decentralized autonomous organization · Legal recognition · Blockchain · Ontology · Business process · B2B

1 Introduction

Traditionally the concept of the contract covers a spoken or written agreement governed by court procedures. The first prerequisite to becoming a legally valid contract is that the contracting parties are engaged voluntarily to reach a consensus. In a traditional contract, a service is offered for some form of compensation (usually money) and some other provisions (e.g., contract terms and conditions, the service delivery dates, liability and compensation for the breach, and so on). Subsequent transactions are based on trust, and contracting parties generally see contracts as a symbol of an existing business deal. Another drawback in traditional way of establishing and managing contracts is that they are often underspecified. More importantly, traditional contracts do not provide sufficient details about the actual process of the transaction and, as a result, frictions between the contracting parties are very likely to occur, e.g., one party assumes a specific product certificate before delivering a partial compensation, and the other party assumes the contrary. The resulting deadlocks result in costly conflict resolution or even the entire contract transaction collapsing. Traditional contract enforcement is also proving to be either too complicated, time-consuming, or impossible, certainly in international circumstances.

Blockchain has established a new type of decentralized autonomous organization (DAO) whose activities run on a peer-to-peer network, involving governance as well as decision-making rules. The latter is an organization whose business provisions are written in a programming code, and the necessary business operations are controlled automatically as per the agreeing to the provisions [25]. DAO encourages re-implementing each aspect of traditional organizational governance, replacing voluntary compliance with a business's agreement with actual compliance using pre-agreed smart-contract code. The latter is machine-readable software code that is situated on the protocol layer of a blockchain system to govern transactions between DAOs [2]. Subsequently, Ethereum [1] emerges as the first smart-contract system where the protocol layer is equipped with a Turing-complete programming language. This innovation affects a growing number of application cases, e.g., in logistics [3], e-healthcare [8], cyber-physical systems [28] such as for smart electrical-grid production [10], and so on. In the meantime, various smart-contract systems exist such as Neo[1], Cardano[2], Hyperledger[3], etc. with varying blockchain types, consensus algorithms, and machine-readable languages [15,29].

By punishing opportunistic behavior, contracts and contract law lead to the enforcement of the intentions initially specified in the contract by the acting parties. Contracts are usually defined as legally-binding agreements that stipulate the rights and obligations of the contracting party towards each other [30]. This study [5] shows when code is law, it refers to the idea that, with the advent of digital technology, code has progressively established itself as the predominant way to regulate the behavior of Internet users. Yet, while computer code can enforce rules more efficiently than legal code, there are also limitations, mostly because of the difficulty to transpose the ambiguity and flexibility of legal rules into a formalized language for interpretation by a machine. A number of studies focus on checking the legality of smart contracts. This article [6] considers the

[1] https://neo.org/ Neo blockchain—Home Page.

[2] https://cardano.org/ Cardano—Home Page.

[3] https://www.hyperledger.org/ Hyperledger—Home Page.

potential issues with legal and practical enforceability that arise from the use of smart contracts within both civil- and common-law jurisdictions.

This paper [13] shows, the technology of smart contracts neglects the fact that people use contracts as social resources to manage their relations. The inflexibility that they introduce, by design, short-circuit a number of social uses to which law is routinely put. Few studies show that smart contracts are a new form of preemptive self-help that should not be discouraged by the legislatures, or courts [23]. A smart contract gives rise to a novel means of legally enforcing obligations and rights. These issues are treated differently from country to country. Smart contracts that underpin transactions in ICOs (Initial Coin Offerings – e.g., KodakCoin) may be illegal in some jurisdictions, while a smart contract that handles intra-institutional banking and other financial transactions is considered as legal, in the same jurisdiction, or elsewhere [22]. Another study show the necessary requirements and design options for the legality of smart-contract forms and proposes future research directions [4]. The future research direction aims to provide straight-through processing of financial contracts, with highly automated smart contract code to entire semantics of smart contract. This future direction opted by the another study [17]. This research shows the significance of highly automated smart contract so called self-aware smart-contract in the real world financial contract. In [22], another initiative investigates the legal enforceability of smart contracts. In this research, the findings render smart contracts legally enforceable by incorporating crypto primitives such as a digital signature.

Thus, the state of the art shows that CCs cause high transaction costs due to their multiple shortcomings, SCs lack legal relevance and this yields legal uncertainties for users while furthermore, SCs are inflexible code that are not smart. This paper fills the gap by answering the main research question of how to establish legal relevance for smart contracts that have socio-technical utility? To establish a separation of concerns, we deduce the following sub-questions. What conditions need to be fulfilled for SCs to have legal relevance? What are the properties of an ontology that represent these conditions for legal relevance? What enactment mechanisms ensure the legal enforceability for contracts?

The remainder of this paper is structured as follows. Section 2 presents a running case and additional preliminaries. Section 3 comprises the legal problem factors for smart contracts that require legal relevance. Next, Sect. 4 translates these elements into an ontology for SCs and Sect. 5 presents the SLC lifecycle model to monitor contractual rights and obligations. Section 6 demonstrates a feasibility evaluation and discussion that expands the running case of this paper. Finally, Sect. 7 gives conclusions, limitations, open issues and future work.

2 Motivating Example and Preliminaries

In Sect. 2.1 we present the running contract case that stems from real car production supply chain contracts. In Sect. 2.2 we present related background literature that prepare the reader for subsequent section.

2.1 Running Case

To illustrate the approach of the paper, a generic supply chain running case of car production is shown in Fig. 1. The original equipment manufacturer (OEM) actually assembles the delivered car parts. The other parties of the supply chain involve either the supply side, or demand side. E.g., the Supplier A sources and supplies the raw materials to Supplier B, who manufactures the individual car components. The particular car component are then shipped to the OEM, who assembles the final product.

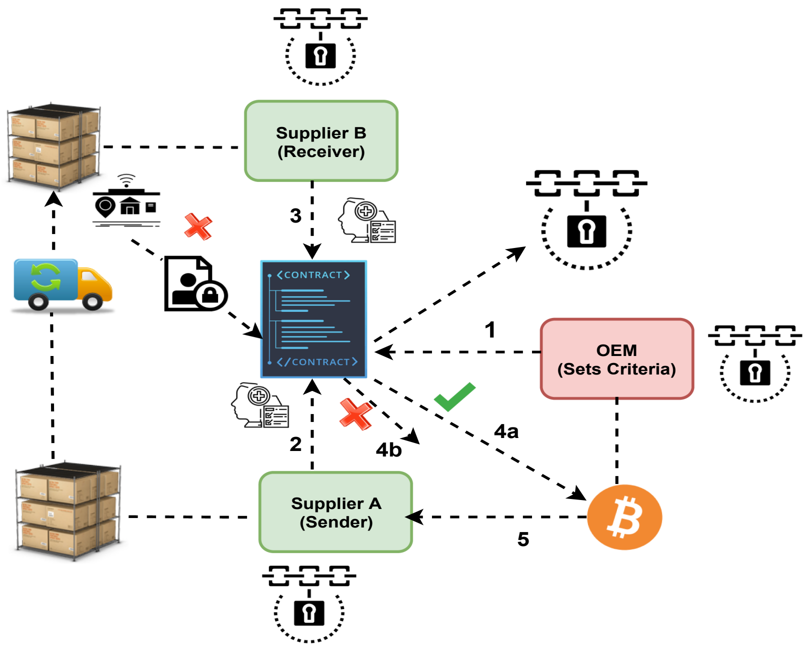


Fig. 1. Supply chain running case.

We deploy the supply chain running case into the blockchain that plays a significant role for checking provenance and tracking of product, which is possible with the integration of smart-contract. The blockchain allows supply-chain partners to access various functions, i.e., partners can access particular function such as checking provenance. For checking the provenance, Supplier A delivers the raw materials to Supplier B and the former publish the records into the blockchain with the integration of sensors. The records include quantity, quality of product and location, time on which the raw product is shipped. This information is immutably stored into the blockchain and can be accessed by supply-chain partners.

For tracking of the product, when Supplier B receives the raw materials, we assume he confirms that the shipment is in order. If the shipment is received on time and at the correct location as per sensor verification, then the payment in the form of digital transaction is executed automatically by a smart-contract.

We can improve supply chain processes efficient by integration of smart-contract into the blockchain. However we introduce various limitation of this technology by considering supply chain running case. In Fig. 1, Step 1 shows the OEM sets the acceptable criteria such as correct location and lead time for each leg of the shipment. The OEM also holds digital currencies while Supplier A is a sender who delivers the raw material and publishes the details on the blockchain in Step 2. When the Supplier B receives the raw material, he publishes the records as well in Step 3. In this step we show some conflict situation that may arise due to immaturity of novel blockchain technology. From supplier B, the data goes to smart contract through sensor for further action. This data can not reliable because it can be altered by external third party. In next phase, when specific criteria meet that are embedded in the smart-contract per Step 4a, then a payment is executed automatically in Step 5. There can be another possibility that the smart contract fails to meet certain conditions. In this situation, an alert is triggered so that partners can rectify any problems in Step 4b. Furthermore, we raise another issue for this paper in asking what happens if a particular obligation is not performed by the partners? For example, assuming the OEM has a payment obligation in Step 5, then in case of late payment, Supplier A has the right to claim late-payment charges. After enabling the corresponding function of rights by Supplier A, the OEM has an obligation to pay. Here, we have seen rights and obligation of parties are not clear. There can be another possibility that the smart contract fails to meet certain conditions. In this situation, an alert is triggered so that partners can rectify any problems in Step 4b. Another challenge of this paper is, if the partner is a non-programmer, he is not capable of understanding what rights and obligation are written in the contract.

2.2 Related Background Literature

In this section, we describe the computation toolkit to understand the solution of the running-case problem. In Sect. 2.1, we propose the situation of the dispute among the parties. To overcome this situation, we develop an ontology that comprises the concept and properties of rights and obligations. The ontology is a formal representation of knowledge by a set of concept and relationship among those concepts [14]. The ontology organizes the class hierarchies of relationships and allows the practitioner to understand the relationships of the particular problem domain. We design the ontology in Protege tool [16] that is open-source ontology editor and comprises the graphical user interface for visualization of the relationship among classes. We employ the HermiT-tool reasoner [7] that checks the correctness of ontology and identify subsumption relationship among classes.

Later, to automate the concept of rights and obligation in smart-contracts, the Protege tool also supports web ontology language[4] (OWL) that express the formal semantics of ontology into machine-readable code. We also employ Coloured Petri Nets (CPN) tools[5] for checking the dynamic behaviour of ontology processing. CPN tool is an software that is useful for simulation and state space analysis of the model. An state space is a directed graph with nodes so-called states and arcs that connect states and transition. The CPN-notation comprises state represented as a circle, transition represented as a rectangle, arcs that connect the states to transitions, and token with color, i.e., attributes with values. Transition fires when all input states hold the tokens and produce condition-adhering tokens into output places.

3 Legal Recognition Factors

In this section, we discuss the legal recognition of the business-to-business (B2B) smart contracts presented in Sect. 2.1. A core principal in contract law is freedom of contract that has two main elements, a) the right of a legal person to freely decide whether to enter into a contract and b) the right to freely decide together with the other contracting party, or parties about the content of the contract [24]. The prevailing view in law [26] concerning a) is that in principal nothing prevents two B2B parties to voluntarily enter into machine-readable sales contracts that are based on a blockchain[6]. To assess how the rules apply regarding the formation of contracts to smart contracts, the analogy of a vending machine has often been used in the literature [27]. In both cases, a legally binding contract is formed due to the consenting actions of the contracting parties. Just as for a vending machine, a smart contract can be independently designed by an offeror and deployed to a blockchain [6]. The design and deployment thereby indicate the offeror's intention to be legally bound according to the terms stipulated in the smart contract. According to this analogy, the offeree agrees to be bound by the smart contract through conclusive conduct. Equivalent to entering a coin into a vending machine he, or she fulfills the triggering requirements of the smart contract. In regard to our running case, this means that an OEM sends cryptocurrencies to Supplier B's smart contract to order intermediate product. By sending the tokens to the smart-contract wallet address, the OEM consents to the contract terms of Supplier B's smart contract and a legally binding agreement is formed [11]. To conclude, from a legal point of view, the right of a legal person to freely decide whether to enter into a contract extends to the right to enter into smart contracts.

[4] https://www.w3.org/OWL/.

[5] http://cpntools.org/.

[6] Note that this freedom may be limited in the case of business-to-consumer contracts and special kind of contracts with significant consequences for the contracting parties (e.g., the selling and transfer of real estate) where the law may demand a special form requirements. Still, as these contracts are not part of our running case, we will not discuss the legal problems connected to the operationalization of such contracts as smart contracts any further.

Accordingly, also b) the right to freely decide together with the other contracting party, or parties about the content of the contract, should extend to smart contracts. Still, it is currently unclear how a court, or an arbitrator would interpret smart contacts that do not use legal terminology but are based on programming code [9]. Assuming that for our running case, Supplier B delivers the intermediate product to the OEM who's sensors accurately recognize the delivery of the intermediate product. The cryptocurrency funds are thus released to Supplier B and the goods are stored in the warehouse. Later, the intermediate product is used for production by the OEM and it is discovered that the intermediate products are of inferior quality, most likely due to impurified raw materials delivered by supplier A to Supplier B. The OEM's production process has to stop causing severe consequential damages. The OEM sues supplier A for replacement of the inferior intermediate products and compensation for the lost production. Supplier A claims that the inferior quality of the intermediate products was not caused by impurified raw materials but rather because of inappropriate storage by the OEM. Furthermore, Supplier A argues that the terms and conditions of the smart contract exclude consequential damages and stipulate an immediate notification deadline in case of defective goods that the OEM has missed. The OEM and supplier A also disagree about the responsible dispute resolution forum and the applicable contract law. The current state of smart contracts do not allow a judge to solve these problems. This is because there is currently no complete legal ontology that allows programmers to design smart contracts in the fashion of traditional contracts.

In the next section, we present a comprehensive contract law ontology that allows contracting parties to precisely define the content of a smart contract for machine readability and in case of disputes, also by a human judge, or arbitrator. We argue that only with the help of our contract-law legally binding smart contracts can be designed.

4 Smart-Legal-Contract Ontology

A smart legal contract (SLC) ontology[7] in Fig. 2, Fig. 5 comprises essential concepts and properties to support SC legally enforceability. We develop the SLC ontology in protége tool to render the latter machine-readable and explain the concepts and properties for the running case of the car production business case, as illustrated in Sect. 2.1.

[7] Full ontology: https://bit.ly/3c5eYO5.

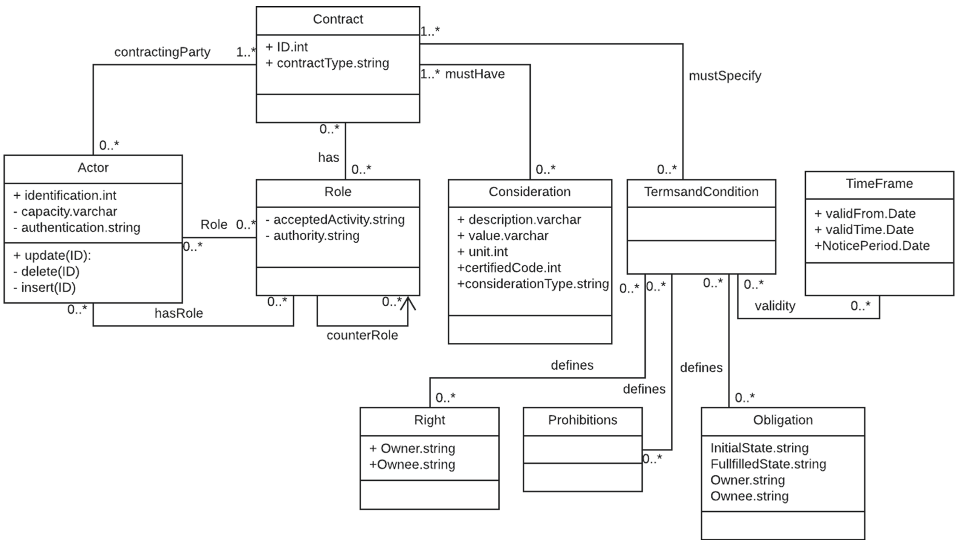


Fig. 2. Outline for upper level smart contract ontology.

The SC ontology defines fundamental concepts such as roles, considerations, rights, and obligations, etc., as presented in Fig. 2. In our running case, Supplier A promises to deliver the raw materials to Supplier B in return a specific amount of money. Supplier B manufactures the car component and promises to deliver to the OEM. To prevent a conflict among the parties the role is defined in the SC. A promise is a statement of commitment to fulfil some act or perform certain deeds and when a promise is given with legal intent of enforcement in court, later becomes legal obligation.

The valid SC must hold two or more contracting parties, such as offeror, offeree, and acceptor etc., [9]. An offeror sends an offer to an offeree for delivering services, and the offeree accepts an offer, offeree becomes acceptor. In our running case, Supplier B acts as an offeror for car components to an OEM that is an offeree. When an offer is accepted by an OEM, then the later transforms into an acceptor. Furthermore, contracting parties must have the necessary competence and capability to perform certain deeds. Supplier B must own the car components as he has promised to deliver the car components, and the OEM must have the capital to pay for it. Further, a valid SC can be a complete-, or incomplete contract. For example, the car manufacturer writes the SC of shipment of cars without giving the deadline, and it considers a valid contract while it is incomplete because there is nothing obligations are specified for the parties (Fig. 3).

Consideration is defined as exchanged value for the trade of which the contracting parties agree to enter into a SC. Car components and raw materials are the asset, or consideration of the contract. The selling of raw material, or car components constitutes the performance that fulfills the promise of the contract. Selling of the car component to an OEM is an obligation of supplier B that is

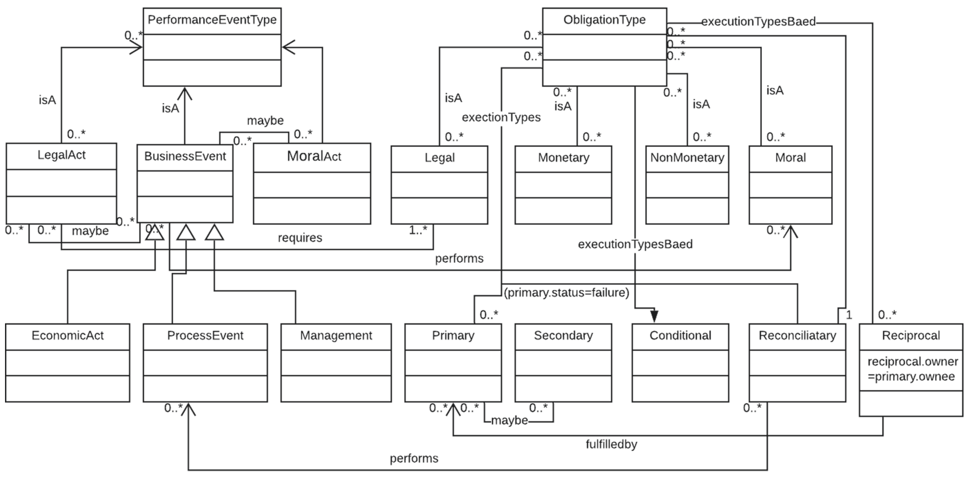


Fig. 3. Rights and obligations.

realized when an actual business act of given car component is performed in return of compensation. An obligation must have obligee who is obliged to perform a particular action and obliger beneficiary who receives the consideration for whom a promise is established. In this case, Supplier B is an obligee who is obliged to deliver car components, and OEM is an obliger who receives the car component.

Also, the SC contains the deadline, or time frame under which the promised performances shall occur. If certain promises are not performed under the deadline, or in unsatisfactory manner, then the state of obligation will change as to unfulfilled. In our running case, if supplier B does not execute the promises as planned and agreed, then the obligation is unfulfilled, resulting in the OEM may seek the pre-agreed rights as compensation. In this case, the OEM may have the right to seek a remedy in the form of a penalty, or can terminate the contract. Also, the OEM can choose not to do anything and settlement occurs by mutual consent.

We explain the simple running case of ontology, where we see an obligation may activate another obligation and rights. Similarly, the rights too may enable new obligations being formed. Therefore, we present the types of rights and obligations in Fig. 5. Also, we describe the change of obligation state under which rights may activate.

We refine an obligation type by adding sub-classes such as monetary-, non-monetary-, moral-, and legal obligation to express a particular remedy. The monetary obligation may create a remedy such as late-payment-charges, penalty, etc. For instance, if the OEM receives a car component from Supplier B and does not pay the money within a deadline, then Supplier B may provide a remedy as to a late-payment-charge. Similarly, if a supplier delivers defective car parts then the OEM may have legal rights to cancel the SC. The OEM may have non-monetary obligations to arrange a carrier to provide the car components but does not have monetary obligation between supplier and OEM. There may be another moral obligation type that is not severely binding but is morally, or ethically an expected obligations. For instance, the OEM has an obligation to pick up the car components from the supplier premises. Still, the OEM may urge for help to arrange the transportation. The supplier may not legally bound to help the OEM, but morally he is bound to assist the OEM (Fig. 4).

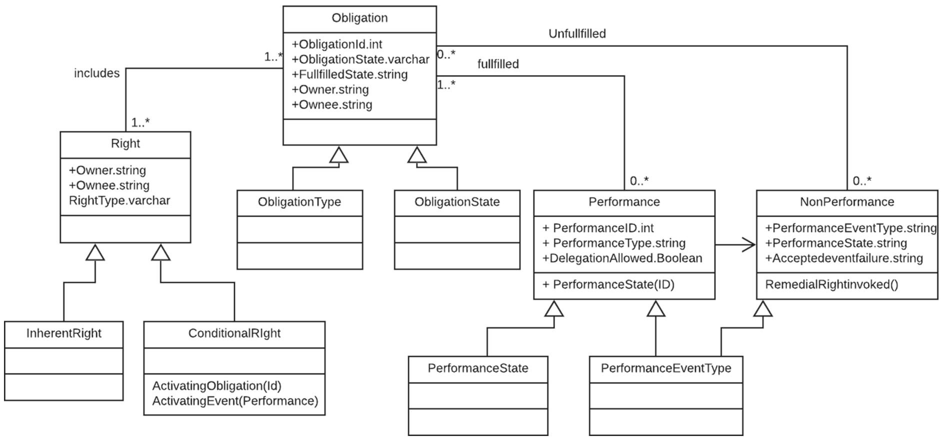


Fig. 4. Common obligation types.

We identify several obligation states to track the SC fulfillment process more systematically, and an individual obligation may exist in more than one category. The proposed obligation states in Fig. 5 are adopted from similar work proposed by LEE [12]. Initially, the SC has an inactive state when the supplier and OEM have signed the SC, but the SC execution does not commence. The SC is said to be active when the performance event is performed. For instance, the OEM demands to deliver the car components to Seller B who receives an order; the supplier obligation to deliver the car components is triggered. The SC obligation state is pending when the supplier has dispatched the car components from the warehouse and is waiting for a carrier for transportation. The obligation state will be pending until the entire essential fulfillment process is completed. When the OEM receives the car components that satisfy the stipulated performance condition, then the obligation state is fulfilled and when the obligation is terminated by the obligee with mutual intent, the obligation state changes to termination.

5 Rights- and Obligations Monitoring

We develop an SLC lifecycle model[8] to monitor contractual rights and obligations defined with the ontology of Sect. 4. We adopt the existing formalized

[8] Full download CPN model: shorturl.at/cxBE9.

smart-contracting lifecycle in [18,20,21], where the startup phase commences with the configuration of a business network model (BNM). The latter is a cross-enterprise collaboration blueprint and contains the legally valid template contract that inserts the service type with organization roles. The latter enables fast and semi-automatic identification of contracting parties for knowing their identity, services, and reputation. We include the rights and obligations throughout the existing smart-contracting lifecycle to monitor the related contractual fulfillment process, and the updated lifecycle is called the *SLC lifecycle model.*

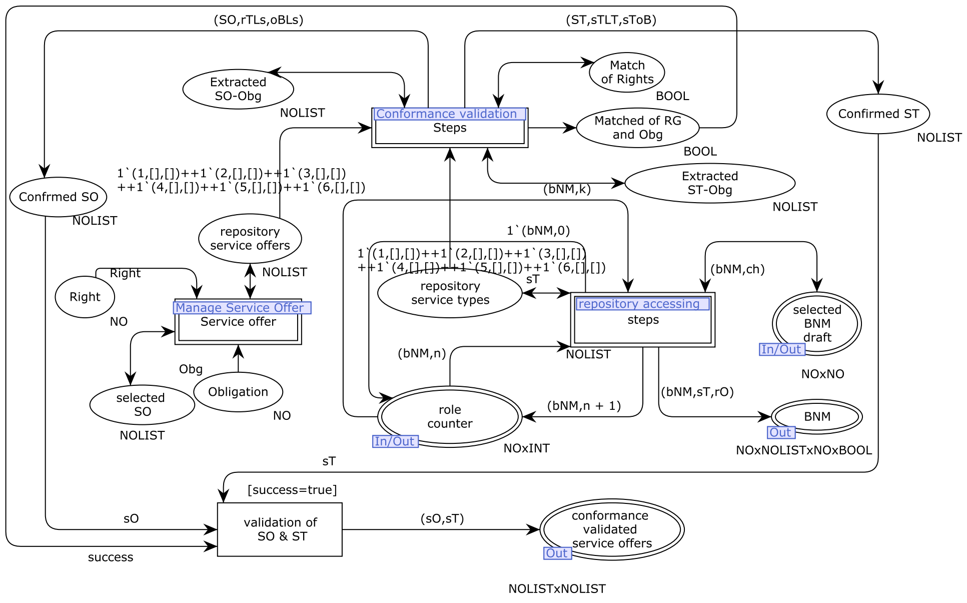


Fig. 5. Rights and obligations selection in BNM.

The SLC lifecycle is divided into two modules, set up- and perform phase. In the *setup module*, the proposal of SC is finalized and negotiated among the contracting parties. Further, the performances of smart- contracting are accomplished in the *perform module*. The rights and obligations are stipulated throughout the entire lifecycle. Still, we present the transaction of rights and obligations in the smart-contracting setup phase and especially in the BNM selection that is an ecosystem for breeding service types with rights, obligations, and roles to become a part of BNM. The latter is divided into several sub-modules namely, *repository accessing*, *manage service offer*, and *manage service type*, as presented in Fig. 7. A *repository accessing* exists in *BNM selection*, where we assume a user inserts the service type with rights and obligations over the time and the same assumptions hold for the repository of service offer in the *manage service offer* module. Finally, the *conformance validation* module is developed to conform to the validation of service offer against the chosen service type in the BNM draft specification.

Next, we provide the details of each module in the subsections below.

5.1 Repository Accessing

We assume a contracting party inserts the rights, roles, and obligation for service types in the repository in Fig. 6.

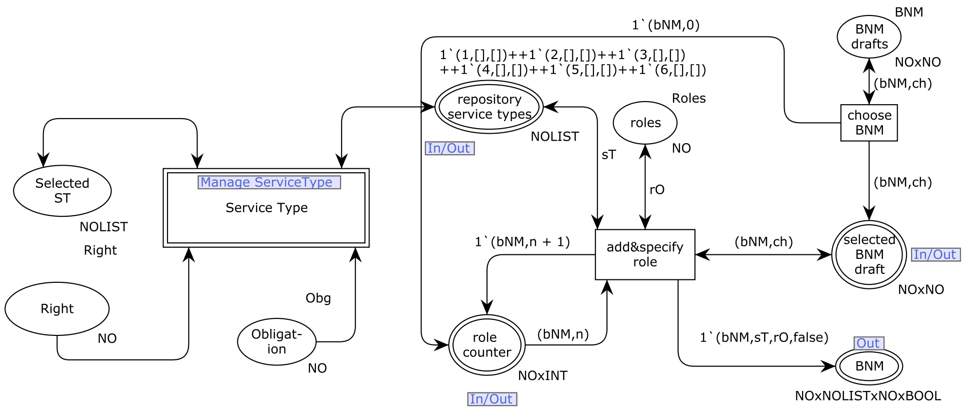


Fig. 6. Repository of service type with rights, roles, and obligations.

As a first step, the contracting party chooses a BNM from stored BNM drafts. Thereafter, the latter inserts the rights and obligations in the *manage service type* module that is stored in the state labeled *repository service types*. Further, the actual BNM selection involves choosing a BNM draft for validating service offers and roles to be filled subsequently with rights and obligations. The rights and obligations to be filled in the *manage service type* module are presented below.

Manage Service Type. The actual repository of service types commences with choosing the rights and obligations in the *Manage service type* module, as presented in Fig. 7. Initially, the list of rights and obligations of service type is empty in the repository. At a first step, the contracting party chooses the service type Id from *choose ST* transition. After that, the latter inserts the rights and obligations simultaneously in *selected ST* by firing the *insert right* and *insert obligation* transitions.

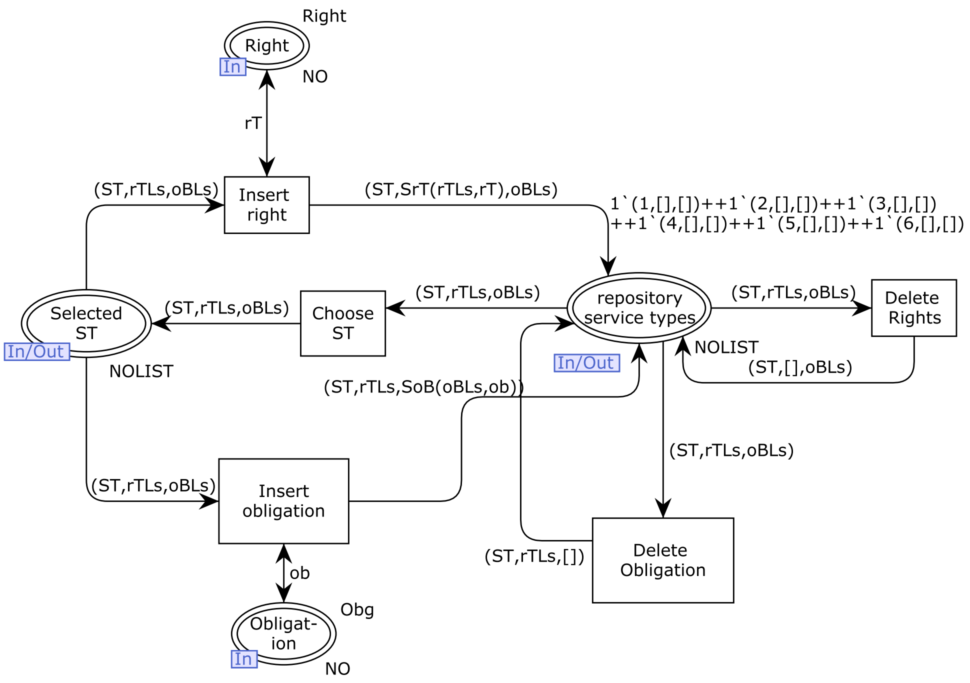


Fig. 7. Insertion, deletion of rights and obligations in service types.

Additionally, the inserted rights and obligations are deleted by firing the transitions *delete right* and *delete obligation*. The same assumption holds for choosing the rights and obligations for service offer in *manage service offer* module.

5.2 Conformance Validation

To be considered as a service offer for finalizing the proto-SC, beforehand, a conformance validation is necessary, as depicted in Fig. 8. The selected service offer and service type from the BNM repository are extracted for the validation. The chosen right and obligation properties inherit the properties of the service type. Thus, we extract the rights and obligations from the state labeled *repository service type* and *repository service offer*. Further, a service offer matched with service types is stored in the state labeled *confirmed SO* and *confirmed ST*.

6 Evaluation

We use the CPN tool to evaluate the model concerning the correctness and performance checking, especially considering aspects that are required for system development. Due to page limitations, we do not present the entire steps that are taken to produce the evaluation results. Several properties, such as reachability, loops, etc., as depicted in Fig. 9, are essential to evaluate in the model. Due to the size of models, computation of states verification through automatic

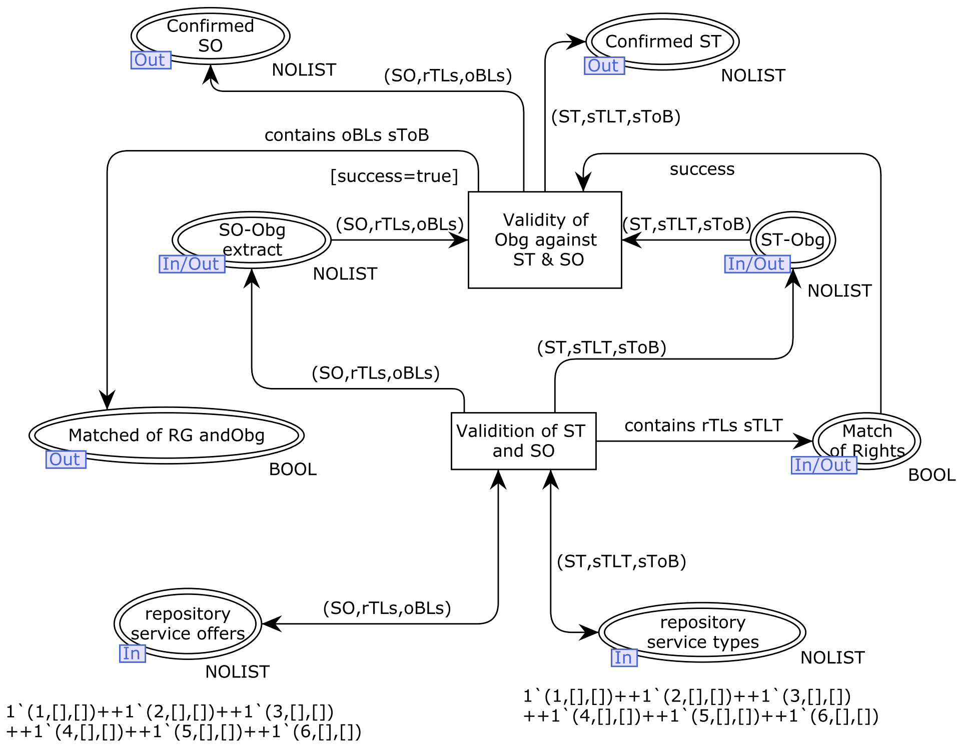
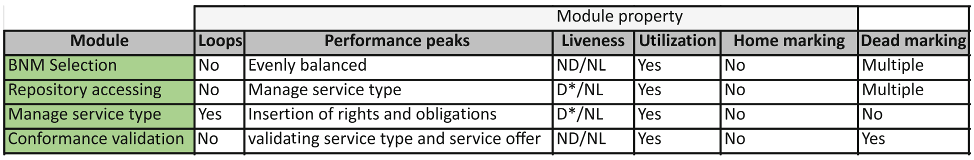


Fig. 8. Conformance validation of service offers and service types.

simulation of token games is challenging. Thus we are focused on the detection of loops to prevent the desired termination reachability. Furthermore, specific attention is required to exit loop-conditions effectively, such as elements of the business-policy control. Performance peaks are calculated during runtime either in designing for sufficient resources or in restricting the load with the business policy control. The utilization property is used to ensure the effectiveness of each model in a specific scenario. Finally, home marking is needed for consistent termination to ensure simple testing of a real system.

We generate the state space on CPN modules where the computation is feasible and present the results in Fig. 9. Loops exist in *manage service type* module. For the Manage service type, a party inserts the rights and obligations.



	Module property					
Module	**Loops**	**Performance peaks**	**Liveness**	**Utilization**	**Home marking**	**Dead marking**
BNM Selection	No	Evenly balanced	ND/NL	Yes	No	Multiple
Repository accessing	No	Manage service type	D*/NL	Yes	No	Multiple
Manage service type	Yes	Insertion of rights and obligations	D*/NL	Yes	No	No
Conformance validation	No	validating service type and service offer	ND/NL	Yes	No	Yes

Fig. 9. Model checking.

Loops exist in *manage service type* module. Managing the service type loop is self-restricting as it only processes the rights and obligations of parties, respectively. The results show for the remaining modules in Fig. 9; they do not contain the loops.

Performance peaks exist in Fig. 9 to represent the places of the SLC lifecycle that are the performance bottlenecks. Peaks exist in each module but not in *BNM selection*. For the repository accessing, the peaks exist for choosing a BNM draft and party's roles and also for inserting the rights and obligations.

There is no home marking, as presented in Fig. 9, and the result for dead marking differ. Multiple dead marking and home marking show test cases are more demanding for practitioners to validate the implementation. D* means a dead marking result that shows the intentional disabling of marking path for the purpose of focusing on a particular module under investigation. Finally, the utilization test in Fig. 9 shows there is no unused sub-module exist.

Due to page limitations, we refer to the reader [19] for more details of evaluation.

7 Conclusion

Smart contracts are machine-readable software code that is situated on the protocol layer of a blockchain system to govern transactions. The later gives rise to a novel means of legally enforcing rights and obligations. State of the art shows that CCs cause high transaction costs due to their multiple shortcomings, SCs lack legal relevance, and this yields legal uncertainties for users while furthermore, SCs are inflexible code that is not smart.

In this paper, we identify a gap between business process management and SC execution, and their fulfillment. Thus, we introduces the ontological concepts of rights and obligations for SC's that are defined in business contracts. For developing the ontology we opt the protege tool that is open source ontology editor and employ the HermiT-tool reasoner that checks the correctness of ontology. Further, we propose the obligation states through which each obligation is passed. For ensuring the legal enforceability of SC's, we present an SLC lifecycle model to monitor contractual rights and obligations. For exploring the SLC lifecycle in a dependable way, we choose CPN Tools that has a modeling notation backed with formal semantics.

The limitation of the paper is that we are only focused on presenting the transaction of rights and obligations in the smart-contracting setup phase. Future work in SC's domain includes analysis and modeling of other types of business legal SC's. Further, resetting of human to the machine for the self-aware SC's are a possible extension to ongoing work.

Acknowledgments. This article is based on research from the Erasmus+ Strategic Partnerships Project - 2018-1-RO01-KA203-049510 "Blockchain for Entrepreneurs - a non-traditional Industry 4.0 curriculum for Higher Education".

References

1. Buterin, V., et al.: Ethereum white paper. Github Repository, pp. 22–23 (2013)
2. Butterin, V.: A next-generation smart contract and decentralized application platform (2014)
3. Casado-Vara, R., González-Briones, A., Prieto, J., Corchado, J.M.: Smart contract for monitoring and control of logistics activities: pharmaceutical utilities case study. In: Graña, M., et al. (eds.) SOCO'18-CISIS'18-ICEUTE'18 2018. AISC, vol. 771, pp. 509–517. Springer, Cham (2019). https://doi.org/10.1007/978-3-319-94120-2_49
4. Clack, C.D., Bakshi, V.A., Braine, L.: Smart contract templates: foundations, design landscape and research directions. arXiv preprint arXiv:1608.00771 (2016)
5. De Filippi, P., Hassan, S.: Blockchain technology as a regulatory technology: from code is law to law is code. arXiv preprint arXiv:1801.02507 (2018)
6. Giancaspro, M.: Is a 'smart contract' really a smart idea? Insights from a legal perspective. Comput. Law Secur. Rev. **33**(6), 825–835 (2017)
7. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: an OWL 2 reasoner. J. Autom. Reason. **53**(3), 245–269 (2014)
8. Griggs, K., Ossipova, O., Kohlios, C.P., Baccarini, A., Howson, E., Hayajneh, T.: Healthcare blockchain system using smart contracts for secure automated remote patient monitoring. J. Med. Syst. **42**(7), 130 (2018)
9. Idelberger, F., Governatori, G., Riveret, R., Sartor, G.: Evaluation of logic-based smart contracts for blockchain systems. In: Alferes, J.J.J., Bertossi, L., Governatori, G., Fodor, P., Roman, D. (eds.) RuleML 2016. LNCS, vol. 9718, pp. 167–183. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42019-6_11
10. Imbault, F., Swiatek, M., De Beaufort, R., Plana, R.: The green blockchain: managing decentralized energy production and consumption. In: 2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe), pp. 1–5. IEEE (2017)
11. Lauslahti, K., Mattila, J., Seppala, T.: Smart contracts-how will blockchain technology affect contractual practices? ETLA Reports (68) (2017)
12. Lee, R.M., Dewitz, S.D.: Facilitating international contracting: AL extensions to EDI. Int. Inf. Syst. **1**(1), 94–123 (1992)
13. Levy, K.E.: Book-smart, not street-smart: blockchain-based smart contracts and the social workings of law. Engag. Sci. Technol. Soc. **3**, 1–15 (2017)
14. Maedche, A., Staab, S.: Ontology learning for the semantic web. IEEE Intell. Syst. **16**(2), 72–79 (2001)
15. Mohanta, B., Panda, S., Jena, D.: An overview of smart contract and use cases in blockchain technology. In: 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pp. 1–4. IEEE (2018)
16. Musen, M.A., et al.: The Protégé project: a look back and a look forward. AI Matters **1**(4), 4 (2015)
17. Norta: Self-aware smart contracts with legal relevance. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2018)
18. Norta, A.: Establishing distributed governance infrastructures for enacting cross-organization collaborations. In: Norta, A., Gaaloul, W., Gangadharan, G.R., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9586, pp. 24–35. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-50539-7_3

19. Norta, A., CINCO, C., Computing, I.: Safeguarding trusted ebusiness transactions of lifecycles for cross-enterprise collaboration. Technical report C-2012-1, Department of Computer Science, University of Helsinki, Helsinki, Finland (2012)
20. Norta, A., Othman, A.B., Taveter, K.: Conflict-resolution lifecycles for governed decentralized autonomous organization collaboration (2015). https://doi.org/10.1145/2846012.2846052
21. Norta, A.: Creation of smart-contracting collaborations for decentralized autonomous organizations. In: Matulevičius, R., Dumas, M. (eds.) BIR 2015. LNBIP, vol. 229, pp. 3–17. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21915-8_1
22. Patel, D., Shah, K., Shanbhag, S., Mistry, V.: Towards legally enforceable smart contracts. In: Chen, S., Wang, H., Zhang, L.-J. (eds.) ICBC 2018. LNCS, vol. 10974, pp. 153–165. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94478-4_11
23. Raskin, M.: The law and legality of smart contracts (2016)
24. Schafer, I.: Ott, lehrbuch der okonomischen analyse des zi-vilrechts, 4 (2005)
25. Singh, M., Kim, S.: Chapter four - blockchain technology for decentralized autonomous organizations. In: Kim, S., Deka, G.C., Zhang, P. (eds.) Role of Blockchain Technology in IoT Applications. Advances in Computers, vol. 115, pp. 115–140. Elsevier (2019). https://doi.org/10.1016/bs.adcom.2019.06.001. https://www.sciencedirect.com/science/article/pii/S0065245819300257
26. Smits, J.M.: Contract law: a comparative introduction
27. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997)
28. Teslya, N.: Industrial socio-cyberphysical system's consumables tokenization for smart contracts in blockchain. In: Abramowicz, W., Paschke, A. (eds.) BIS 2018. LNBIP, vol. 339, pp. 344–355. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-04849-5_31
29. Wang, S., Yuan, Y., Wang, X., Li, J., Qin, R., Wang, F.: An overview of smart contract: architecture, applications, and future trends. In: 2018 IEEE Intelligent Vehicles Symposium (IV), pp. 108–113. IEEE (2018)
30. Wulf, A.J.: Institutional competition of optional codes in European contract law. Eur. J. Law Econ. **38**(1), 139–162 (2014)

Appendix 4

IV

V. Dwivedi and A. Norta. Auto-generation of smart contracts from a domain-specific xml-based language. In S. C. Satapathy, P. Peer, J. Tang, V. Bhateja, and A. Ghosh, editors, *Intelligent Data Engineering and Analytics*, pages 549–564, Singapore, 2022. Springer Singapore

Auto-Generation of Smart Contracts from a Domain-Specific XML-Based Language

Preprint · July 2021

DOI: 10.13140/RG.2.2.34511.61609

Vimal Dwivedi
Tallinn University of Technology
17 PUBLICATIONS **23** CITATIONS
SEE PROFILE

Alex Norta
Tallinn University of Technology
145 PUBLICATIONS **1,207** CITATIONS
SEE PROFILE

Auto-Generation of Smart Contracts from a Domain-Specific XML-Based Language

Vimal Dwivedi[0000−0001−9177−8341] and Alex Norta[0000−0003−0593−8244]

Tallinn University of Technology, Akadeemia tee 15 a, Tallinn, Estonia
`vimal.dwivedi@taltech.ee`
`alex.norta.phd@ieee.org`

Abstract. Smart Contracts are a means of facilitating, verifying and enforcing digital agreements. Blockchain technology, which includes an inherent consensus mechanism and programming languages, enables the concept of smart contracts. However, smart contracts written in an existing language, such as Solidity, Vyper, and others, are difficult for domain stakeholders and programmers to understand in order to develop code efficiently and without error, owing to a conceptual gap between the contractual provisions and the respective code. Our study addresses the problem by creating smart legal contract markup language (SLCML), an XML-based smart-contract language with pattern and transformation rules that automatically convert XML code to the Solidity language. In particular, we develop an XML schema (SLCML schema) that is used to instantiate any type of business contract understandable to IT and non-IT practitioners and is processed by computers. We advocate a pattern, as well as its transformation rules, for converting contracts described in SLCML to smart contracts written in Solidity, a smart contract-specific programming language, in order to reduce the effort and risk associated with smart contract development. We demonstrate and evaluate our SLCML and transformation approach through the specification of a real life-inspired Sale-of-Goods contract.

Keywords: Blockchain· smart contract· decentralized autonomous organization· SLCML· supply chain.

1 Introduction

Blockchain technology has gained traction in a variety of industries, including finance [24] and healthcare [17], due to its distributed, decentralized, and immutable ledger, in which individual entities may not be trusted. Blockchains overcome the intermediary trusted authority by securing and validating a transaction through cryptographic signature and a consensus mechanism. Several blockchains, including Ethereum and Hyperledger, use smart contracts to define business rules and automate business processes that govern transactions. According to Nick Szabo "A smart contract is a computer program or a transaction protocol which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement [31]." Prior to the advent of blockchain technology, Szabo pioneered the notion of smart contracts in 1996. The first version of blockchain (also known as blockchain 1.0) was implemented in 2008 as a byproduct of bitcoin without the capability of smart contracts [13]. Blockchain 2.0 introduces smart contract languages (SCLs) like Solidity, Vyper, and others, which have significantly increased the use of smart contracts and blockchain implementations outside of digital currencies [22].

A smart contract is frequently confused with a computer programme written by an IT programmer, but it is an interdisciplinary concept that includes, but is not restricted to, business, financial services, and legal principles [27]. A smart contract defines how exchanges and disbursements between various wallets in the business and finance domains are shared. A contract is a legal agreement between collaborating stakeholders that consists of consensual commitments in commercial contracts, whereas a smart contract is one in which the commitments are encoded in computer programmes that are automatically executed [27]. Because smart contracts are interdisciplinary in nature, different practitioners such as lawyers, computer engineers, business and finance experts, and others from various domains can collaborate to design, propose, and implement smart contracts. Existing SCLs (such as Solidity [9], Vyper [4], and others) are primarily implemented technologically, and smart contracts written in these languages are incomprehensible to professionals outside the IT sector. Legal properties (rights and obligations, for example) of smart contracts are equivalent to software requirements that the program must meet for the programmer. As a result, legal knowledge is required for IT programmers to write contract content and communicate with business people in order to elicit and clearly define software requirements. Due to a lack of legal knowledge among IT professionals, verifying legal requirements in smart contracts is difficult and time-consuming. The other open problem is existing SCLs are not feasible to formulate complex collaborative business contracts (such as DAOs) in a legally-relevant way [10].

Several workarounds for developing legally binding SCLs have been published in the scientific literature, including SmaCoNat [28], ADICO [16], and SPESC [19]. The above-mentioned publications, in particular, encompass intriguing approaches and findings. However, there is no model transformation to an executable smart contract implementation in the proposed domain specific languages [12]. Furthermore, existing SLCs are not process aware when it comes to writing collaborative business contracts. Therefore, this paper fills the gap by answering the research question, i.e., How to build a BPMN choreography model for converting an SLCML contract to Solidity. The paper's contributions include the creation of an SLCML [1] (XML-based language) smart-contract implementation that is process aware and understandable by both IT and non-IT practitioners. SLCML allows for the specification of a smart contract's configuration (rather than its execution) for the purpose of creating collaborative business contracts. To reduce the effort and risks associated with smart-contract development, we propose a pattern and its transformation rules for building a choreography model to translate smart contracts written in SLCML into Solidity. We derive the following sub-research question from the main research question in order to simplify it and develop a separation of concerns. What is the structure of the SLCML instantiation that is crucial for the choreography transformation? What are the patterns and rules for converting SLCML code to a BPMN choreography model? What is the feasibility-evaluation approach of the proposed solution for a use case?

The following is the outline of the paper. The traceability of rights and obligations in the milk supply chain is discussed in Section 2. We also explain the preliminaries, which help the reader understand the sections that follow. We define the running case SLCML instantiation and present the syntax and structure of SLCML in Section 3. In addition, in Section 4, the pattern and transformation rules are discussed, and the feasibility is evaluated in Section 5. Section 6 describe related work, and Section 7 concludes the paper and provide the future work.

2 Motivating Example and Preliminaries

To demonstrate how legally binding smart contracts can be created, we address an ongoing case study from the dairy food supply chain. We believe that both upstream (manufacturers, producers, etc.) and downstream (distributors, wholesalers, etc.) supply-chain representatives track and process conformance data in order to explain enforcement criteria to both public workers and more demanding consumers. As a result, in Section 2.1, we present the running case and discuss a conflict scenario involving the rights and obligations of upstream and downstream parties. Section 2.2 then describes the related background literature, preparing the reader for the following sections.

[1] shorturl.at/uBHR6

2.1 Running Case

Blockchain technology has the prospects to benefit the food supply chain, including the pork supply chain [8], the fish supply chain [20], and others.A significant use-case for blockchain is the tracking and monitoring of product safety and regulatory compliance throughout the food supply chain [6]. To better understand the traceability of rights and obligations in the context of the various food supply-chain stakeholders, we use the dairy supply-chain conceptual model [2] for our research, as shown in Figure 1. Many stakeholders, including manufacturers, retailers, and others, are in charge of managing the supply-chain operation from the start, when a cow on a farm produces raw milk, to the finished product, when a consumer consumes baby-milk powder. The traceability of one of the actors' internal processes is referred to as internal traceability, whereas chain traceability corresponds to the traceability of the entire supply chain [23]. An external traceability is used to determine the traceability between two actors. Each actor uses a different type of technology, such as IoT devices, location-based technology, to retrieve and provide information to the Food Safety Information System (FSIS). The latter includes a variety of data that are required for food supply-chain actors to achieve transparency and quality assurance. According to [2], FSIS is managed by centerlized-, or decenterlized information that is not specified. The emphasis is on each actor possessing similar abilities at the same time. The Food Safety and Quality Assurance System (FSQAS) defines the safety and quality standards that supply-chain stakeholders must follow. The FSIS stores traceability data that reflects compliance with these regulations.

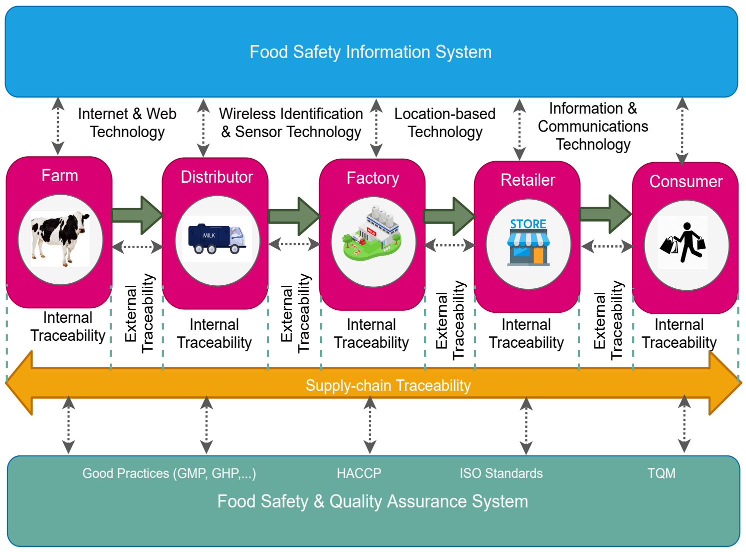


Fig. 1: Dairy food supply chain [3].

We are only interested in market exchanges between entities. Farmers keep detailed records of their farm's location, breed, immunisations, treatments, and, if applicable, special regimens. Animal health and movement are tracked using RFID devices, or any other sensor network that incorporates blockchain technology. Similarly, data on animal migration is captured and stored on blockchains using sophisticated machines. When the milk is milked, the distributor is notified via public blockchain platforms, and the milk is ready for collection. Maintaining temperature during transportation is critical to preventing milk spoilage, and sensors devices are used to accomplish this. Furthermore, GPS is used for real-time vehicle tracking. As milk is delivered to the factory, the relevant information is updated on a blockchain network. The location of the unit, the amount of delivery at a specific lot, and so on are examples of information. The factory processes the milk and produces the baby milk powder, as well as providing consumers with accurate information about food products, management guidelines, validity periods, usage directions, and other useful information.

According to [7], smart contracts are required for food supply-chain operations to excel in the form of improved buyer service and quality assurance. The supply-chain operation clauses are specified in the food-safety- and quality-assurance system to trigger specific events. For example, if a distributor fails to deliver milk to producers (i.e., a factory) within a specified time and quality, smart contracts charge a penalty prior to delivery. In a traditional supply chain, collaborating entities frequently have little, or no control over which entities are responsible for bottlenecks. This oversight is made possible by smart contracts and blockchain technology, which allow collaborative parties to monitor and track the status of products and transactions. Still, we raise legal- and business concerns resulting from SCLs and the blockchain technology's infancy. Assuming a smart contract automatically releases funds (ether, bitcoin, etc.) after delivering milk to producers; what happens if the quality of the milk delivered does not meet the producers' specified requirements? When the milk quality is poor prior to delivery, the producer seeks compensation, or exchanges the milk. On the distributor side, the obligation to fulfill that compensation must be imposed. The distributor can also claim that the poor-quality milk stems from the farmer. Legal constraints must be specified in smart contracts in these cases according to contract law. Contracts must include exchange provisions that define the rules for exchanging the product, canceling the contract, and calculating interest adjustments in payments.

2.2 Preliminaries

We discuss the difficulties in writing collaborative smart contracts for the dairy supply chain in the previous section, where parties' rights and obligations must be specified. We use the Liquid studio tool to create SLCML (i.e., XML schema) as a foundation for instantiating each type of real-world business contract. Liquid studio[2] offers robust set of tools for XML and JSON creation, as well as data mapping and transformation tools (such as XSLT-, XQuery editor, and so on), as well as a graphical XML schema editor for visualising, authoring, and manipulating complex XML schemata. The former provides an interactive logical view of the XML schema, allowing for intuitive editing while still retaining access to all aspects of the W3C XML schema standard. Following that, a Solidity code-generator tool is created to convert XML-based smart contracts into blockchain-specific programming languages (i.e., Solidity [1]).

Following that, we next present the SLCML schema definition and SLCML instantiation for legally binding dairy-milk supply chain with the XML smart-contract code.

3 SLCML: A CONTRACT-SPECIFICATION LANGUAGE

SLCML [11] is a machine-readable XML-based choreography language for specifying cross-organizational business smart contracts derived from a smart-contract ontology[3], which includes the concepts and properties of legally binding contractual business collaboration. We do not go into detail about the smart-contract ontology because it is beyond the scope of this paper.

SLCML is an extension of the eSourcing Markup Language (eSML), with the goal of incorporating a smart-contract collaboration configuration. eSML is based on a

[2] https://www.liquid-technologies.com/xml-studioLiquid Studio — Home
[3] shorturl.at/gxFKT

real-world contractual framework, and collaborating stakeholders use process views that are projected externally for cross-organizational matching. When a match takes place an agreement is formed, which is the primary criterion for contract formation. A process view is an abstract concept of an inbuilt process that enables customers to control the evolution of process instances while concealing sensitive or insignificant aspects of the supplier process. Legally binding constructs, as well as process views and their efficient matching relationships, are critical factors in establishing cross-organizational smart contract collaboration. We do not go into great detail about process views and their matching relationships in SLCML instantiation because they are a part of eSML and have already been discussed in [25], [14]. We use a subset of eSML as the base language for SLCML, which we supplement with additional schemas for rights and obligations and the complete schema along with process views can be downloaded from the link provided in Section 1. Following that, we discuss below the SLCML instantiation in terms of rights and obligations that is not part of eSML for our running case, which is based on the SLCML schema.

The code excerpt in Listing 1.1 defines the fundamental contractual elements required for any legally binding business-oriented smart contract. To resolve the conflict, a smart contract is established between the producer (i.e., factory) and the distributor, which has a unique ID and cannot be changed during contract enforcement. Line 2 specifies the producer's public key, and Line 6 specifies the milk distributor's public key. Lines 3 and 7 specify the names of the parties, namely the producer and distributor. The contracting parties' roles i.e., producer as a service consumer and distributor as a milk supplier are defined in Lines 4 and 8, respectively. Line 10 defines consideration of contract (i.e., milk), for which the parties agree to enter into a contract. Next, terms and conditions include the obligations and rights that are defined in Listing 1.2 and 1.3 respectively.

```
1  <contract contract_id="Id1">
2          <party address="03 m6">
3              <name> Producer</name>
4              <role> Service consumer </role>
5          </party>
6          <party address="31 x7">
7              <name> Distributor </name>
8              <role> Milk supplier </role>
9          </party>
10         <consideration> Milk </consideration>
11             <terms_and_conditions/>
12         <obligation/>
13         <right/>
14         <prohibitions/>
15         <terms_and_conditions>
16     </contract>
```

Listing 1.1: Contract instantiation for the dairy supply chain.

Listing 1.2 is an example of a producer obligation to compensate for milk. The obligation has a name and a unique ID that is used to track performance that we consider a monetary obligation because it deals with economic-, or financial consequences. Line 3 activates the obligation state, which means that the producer receives milk according to the orders and has an active obligation to pay money to the distributor. The producer is the obligor who is required to perform this obligation as stated in Line 6. The distributor is the beneficiary of the obligations stated in Line 5, and we assume no third parties, or mediators are involved in this obligation. The to-do obligation has legal consequences, and the producer is required to act by paying the money. Line 12 presuppose the obligations, for which the producer and distributor sign contracts (Act 1) and the producer receives milk. The payment that must be transferred from the producer's wallet address to the distributor's wallet address is referred to as the performance type. In addition, the performance object is defined as a buy with qualifiers that is paid for a specific amount within a specific time frame.

The payment time limit is specified in the rule conditions, and the purchase-payment plan is specified in Line 15. Finally, a reference to the existence of a late payment remedy is added to the obligation. If the producer fails to pay the money within the specified time frame, the producer must transfer a specified monetary amount to the distributor.

```
1  <obligation_rule tag_name ="paying_invoices" rule_id ="0001"
2  changeable ="false" monetary ="true">
3  <state> enabled </state>
4  <parties>
5     <beneficiary> Distributor (31 x7 ) </beneficiary>
6     <obligor> Producer (03 m6 ) </obligor>
7     <third_party> nil </third_party>
8  </parties>
9  <obligation_type>
10    <legal_obligation> to-do </legal_obligation>
11 </obligation_type>
12 <precondition> act1 (signed)& Milk (transferred) </precondition>
13 <performance_type> payment (03 m6,31 x7, buy) </performance_type>
14 <performance_object> invoice ( buy, amount)<performance_object>
15 <rule_conditions> date ( before delivery of milk) </
       rule_conditions>
16 <remedy>late_payment_interest (amount,03 m6 ,31 x7)</remedy>
17 </obligation_rule>
```

Listing 1.2: Obligation example for paying milk.

Listing 1.3 code extract includes provisions that intersect with the obligation. The rights and obligations are intertwined, which means if one party asserts its rights, the other party is obligated to comply. The rights, such as the obligation in Listing 1.2, have a beneficiary who can benefit from them and an obligor who can enable them. For example, if a producer receives poor-quality milk, he or she has the right to demand that the milk be replaced. As a result, the distributor is required to replace the milk.

```
1  <right_rule tag_name ="milk_replacement" rule_id ="0002"
2  changeable ="true" monetary ="false">
3  <state> enabled </state>
4  <parties>
5     <beneficiary> producer (31 x7) </beneficiary>
6     <obligor> distributor (03 m6) </obligor>
7     <third_party> nil </third_party>
8  </parties >
9  <right_type>
10    <conditional_right> claim </conditional_right>
11 </right_type>
12 <precondition> act1 (signed)& Milk (transferred)</precondition>
13 <performance_type> replace (poor-quality milk) </performance_type
       >
14 <action_object> milk (cans of milk, type, and batch unit)</
       action_object>
15 <rule_conditions> deadline (date) </rule_conditions>
16 <remedy> late_replacement_interest (amount, 31 x7) </remedy>
17 </right_rule>
18
```

Listing 1.3: Right example for replacing a poor-quality milk.

Again, we assume that the rights have a name and an ID as defined in Line 1. Because the distributor has the right to revoke the right, the distributor, for example, can persuade the producer that the quality of the milk was ruined during logistics due to a faulty sensor machine that was not his fault. If the distributor agrees to replace the milk, the rights to the contract can be changed while it is being carried out, and the compensation can be placed to false. The parties are defined in the same way as in Listing 1.2, and the state of right is available for immediate enactment. The right-type is set to conditional-right, and the producer asserts that the milk be replaced. For the right to be exercised, the contract must be signed and the milk must be delivered to the producer. To replace the milk described as a performance object, the performance type has been changed to cans of milk, type, and batch unit. Following the activation of this right, the distributor's corresponding obligation must be met within the timeframe specified; otherwise, the producer is entitled to monetary compensation.

In the following section, we discuss the rules for converting SLCML code to a choreography model and then implementing solidity smart contract code.

4 Patterns and transformation rules

We borrow a concept from the translation of ADICO statements to solidity [15] and XML to choreography model [5]. According to Frantz et al. [15], contract statements are divided into different components (abbreviated as ADICO), which include 'Attributes', 'Denotic', 'AIm', 'Conditions', and 'Or else, where attributes denotes actor characteristics and denotic describes obligations, permissions, or prohibitions. AIm describes the action taken to regulate the contract, conditions describe the contract's contextual conditions, and Or-else describe the consequences. Furthermore, the author proposes mapping rules that enable developers to generate solidity code from ADICO components. Our main contribution is to first transform the rights and obligations written in SLCML into choreography model based on the publication [5] and then to solidity code based on the publication [15]. The proposed SLCML-Solidity mapping is summarized in Table 1.

This core construct mapping serves as the cornerstone for translating SLCML specifications into Solidity contracts. Supply chain is a process choreography model referred to as Supply chain choreography model, which is then transformed into a smart contract referred to as "Supply chain smart contract" (rule (a)). External functions are only called externally by other smart contracts, or they can be called if the former includes interaction with other contracts. In our case, interactions occur between the main smart contract "Supply chain smart contract"and the "Supply chain oracle contract", which are discussed further below.

Product quantity, quality, and other constraints are attached to attributes components, which are contract global variables that translate to Solidity struct members (rule(c)). Similarly, performance types effectively represent functions and events (rule(f)), whereas function modifiers introduce descriptive checks that invalidate function execution (rule(d, e, g)) to reflect the mix of rights, obligations, and corresponding preconditions.The performance type (rule(f)) is refined further by enabling the configuration of an item, such as an invoice, as shown in Listing 1.2, and a target related with a particular operation, such as replacement, as shown in Listing 1.3. Events that are prompted as a result of the fulfilment of encapsulated circumstances are a subset of this type (e.g. reaching a deadline for pay). Remedy are repercussions for breaching provisions in function modifiers, which are translated by default utilizing the throw primitive (rule(h)). Conditions joined by quantifiers are defined by a single modifier construct, with semantic integration delegated to the developer. Assume a payment is made, and the payment is represented as a choreography task that interacts with the merchant account, which is implemented in Solidity as an external function.

In the process choreography model, tasks represent steps in the supply chain that interact with external resources. As a result, we discover: 1) information recorded from an external actor (eventually the service providers) and passed to the smart contract, for example, service providers provide data regarding their service costs, and so on, which is to be processed with the implementation of the process choreography model.2) Data collected from external applications and utilities, such as information pertaining to a payment task or configurations of transportation will be similarly passed to the smart contract. 3) data read from smart contracts stored in the

Table 1: SLCML to Solidity transformation rules.

Rule's identifier	XML element	Choreography element	Solidity Code
(a)	Root element: supply chain	Supply chain: choreography model	Supply chain: smart contract
(b)	Step containing a supply chain	Choreography task	External function
(c)	Attributes	Data perspectives	Struct
(d)	Obligation	Choreography task	Function modifier, Events
(e)	Precondition	Choreography task	Function modifier
(f)	Performance type	Choreography task	Functions, Events
(g)	Right	Choreography task	Function modifier
(h)	Remedy	Embedded in model documentation	Throw statements/alternative control flow
(i)	Step containing payment	choreography task	External function

blockchain. Special contracts known as oracles will be used to deal with external data. Oracles are real-world data stores that act as a bridge among smart contracts and the rest of the world, as smart contracts are unable to directly call external programmes.

In the following section, we will use these transformation rules on SLCML code to generate the choreography model and then the solidity code.

5 Feasibility evaluation

Our starting point is the SLCML code corresponding to our running case generated in Listing 1.1, 1.2, 1.3. The transformation rules XML to choreography in Table 1 are first used to generate the process choreography model in Figure 2 and 3 in which two organizations, namely, service consumer and service provider, are engaged in the execution of the cross-organizational milk supply chain process. A service provider organization (e.g., milk distributor) completes a workflow process on behalf of a service consumer (e.g., milk producer). Nonetheless, the service provider does not wish to disclose all of the details of the workflow process that it implements, preferring to disclose only those aspects of the process that are of interest to potential consumer organizations. We do not represent the process views in the SLCML code due to page constraints; however, the complete schema with process views can be downloaded using the link provided in the Section 1. We employ the BPMN notation [30] to visualize the supply chain processes specified in SLCML. An activity can appear in both a consumer's process-view request and a service provider's process-view offer; we distinguish activity occurrences by appending activity labels with either a c: (for a consumer) or a p: (for a service provider) namespace marker (for providers).

Figure 2 depicts a milk producer process view that begins with taking orders from customers and ends with the sale of milk powder. Following the confirmation of the agreement between the producer and the buyer, i.e., the retailer, a signed contract is sent to the client (i.e., the retailer), along with an estimated delivery date. The milk producer would then place an order for milk packaging through its own in-house process and allocate a batch unit. Following that, the supplier outsources a distribution mechanism and specifies the rights and responsibilities of the service

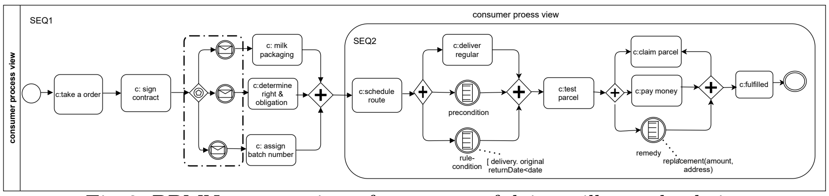


Fig. 2: BPMN process view of consumer of dairy milk supply chain.

provider. To deliver the milk-package, a subprocess (compound node) is executed, which includes the following tasks. First, a route is planned. Following that, the milk-package is shipped with a precondition (i.e., the quality and quantity of milk) and a rule condition (i.e., delivery date). Eventually, the customer is handed the milk package and asked to approve the receipt. If the precondition and rule condition do not fit as defined in the smart contract, the consumer may make a claim for the parcel and request a replacement product. The private mechanism of the service provider in Figure 3 differs from the customer view in Figure 2 in that it includes an event-based gateway through which the service consumer selects which component to follow throughout execution. This decision is depicted by the two message-flows in Figure 3 from the service consumer's in-house domain. As opposed to the customer view, the provider process includes the additional activities p:determine transportation and p:determine route. During implementation, these operations must become invisible to the service user.

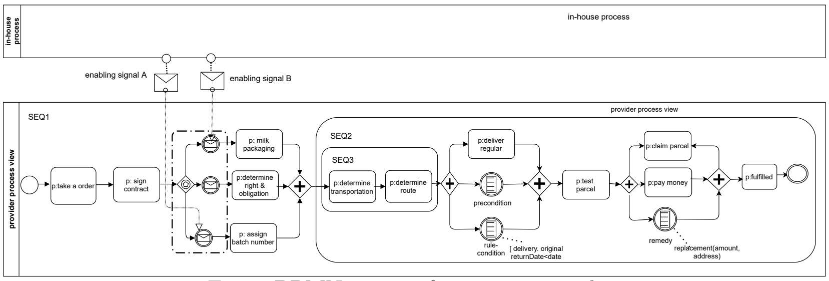


Fig. 3: BPMN process for service provider.

Following that, we will discuss the next component of the transformation rules, namely, choreography to Solidity smart contract as shown in Table 1, which are required to transition from the process choreography model to the Solidity code. As a demonstration, we execute and enact our "milk powder contract" in Caterpillar [26], an open-source Blockchain-based BPM system that converts business processes modelled in BPMN into smart contracts written in Solidity language. Listing 1.4 contains an excerpt from the generated smart contract. To begin the task execution with rights and obligations, our smart contract, "milk powder SupplyChain," contains two events and four solidity functions relying on the transformation rules. Lines 3 to 8 of Listing1.4 represent global variables, and data pertaining to the process state is stored on-chain. As defined in lines 10 to 15, the list of producer and distributor variables is declared in struct, which can be accessed with a single pointer name throughout the contract. In Line 16, a further event for performance type i.e., MilkSupply, is implemented, containing parameters such as milk quantity, producer address, and distributor address, which track the delivery of supply. Lines 17 to 20 implement the notifyObligationBreach event and associated function for tracing the obligations. Similarly, an event for rights is introduced in lines 23-25 in the event that a party seeks compensation. Following that, a modifier precondition is used to release the product if payment is received before the deadline.

```
 pragma solidity ^0.4.16;
 contract milk powder_SupplyChain{
     uint public role;
     address producer;
     uint funds;
    uint milk_quantity;
    uint milk_quality;
    uint public consideration;
    .....
      struct Producer{
         address producer;
         uint role; }
     struct Distributor{
         address distributor;
         uint role;  }
event MilkSupply (uint milk_quanity, address distributor, address
     producer);
event notifyObligationBreach (*Define Type* obligaton, address
    contract );
function notify(*Define Type* obligaton, address producer){
    //TODO: Implement code to notify obligation breach for target
     contract address
    notifyObligationBreach(obligation type, contract);}
 function release(uint milk_quanity, address producer ){
     // TOD: Implement code to relase milk to the producer. }
 event claimParcel(*Define type* right, address contract);
 function replace_parcel(*Define type* right, address contract){
     //TODO: Implement code to activate right for target contract
     address }
    modifier precondition(){
         //Check the condition
         uint benificiary;
         uint obligor;
         if(!paybeforedeadline){
             release(milk_quantity, producer);
             producer.send(funds); }
         else
         {  _;  } } }
```

Listing 1.4: Milk supply chain.

6 Related work

Existing SCLs, such as Solidity and Serpent, are developed from an IT standpoint, with the programmer writing machine-readable code without knowledge of the contract domain. Existing research continues to focus on the development of SCLs to specify legally binding smart contracts, but the majority of research is proposed for specification rather than implementation for blockchains, and no translation mechanism to blockchain specific languages is proposed.

For the purpose of collaborative design, researchers propose a specification language (SPESC) for defining the configuration of a smart contract (instead of its development) in [18]. In SPESC, smart contracts are defined as a combination of IT experts, domain practitioners, and business or financial transactions. Real-world contract utilities such as the role of the party, the set of terms and conditions, and so on can be specified in smart contracts using SPESC. Nonetheless, SPESC does not address many aspects of contracts, such as obligation states, categories of rights and obligations, and so on, but instead focuses on modeling legal relations (legal positions).

The researcher addresses the challenges of formalizing natural-language contracts in machine-readable languages in [32]. Furthermore, the Contract Modelling Language (CML) is proposed for modeling and specifying unstructured legal contracts that cover a wide range of common contract situations. CML specifies a natural-language comparable clause grammar that is similar to real-world contracts; however, this research does not address transaction rules and is insufficient to formalize any type of business contract (viz. domain completeness).

According to researchers [29], human contract intentions are mostly defined in natural language, which is simple to understand but highly ambiguous and open to interpretation. Furthermore, a methodology for creating a high-level specification that achieves common understanding through natural-language phrases and is compiled directly into machine instructions is proposed. Nonetheless, this study focuses primarily on the readability and safety of smart contracts and does not express the domain's collaborative contractual suitability and completeness. This research [21] creates a framework for dynamically binding parties to collaborative process roles as well as an appropriate language for binding policy specifications. The proposed language includes Petri-net semantics, which allows policy consistency to be verified. In this study [5], the transformation rules from XML to process choreography model are proposed first, followed by Solidity code. Nonetheless, the proposed approach is designed specifically for tourist itineraries. This approach is used in our work by extending the transformation rules for generating any type of domain specific smart contract written in SLCML to solidity.

According to the related work, there is a lot of research being done in the area of legal smart-contract specification. Nonetheless, we address the gap that the solutions address in an immature manner, revealing that existing methodologies are restricted to the design of all types of real-world contracts. Prior research, for example, is insufficient to specify collaborative- and legally binding smart contracts. We bridge this gap by adopting a reality-based concept of a legal contract that requires collaborating parties to reach an agreement. This agreement in SLCML represents the matching of process views by a service consumer and service provider, which is fundamentally different from the related work listed with a purely technical focus. In addition, we provide the rules for translating any type of smart contract written in SLCML to Solidity.

7 Conclusion

This paper introduces the XML-based SLCML choreography language for specifying legally binding smart contracts for cross-organizational business collaboration. SLCML is the result of a case study-based investigation, which implies that the language contains vital collaboration constructs as well as a framework for conceptual accuracy. We extend our eSML (previous work), which is built on a real-world contracting foundation; collaborating parties use process views that they project externally for cross-organizational matching. eSML, however, lacks the legally binding construct that is required for the formalization of business contracts. Furthermore, transformation rules are proposed to reduce the effort and risk associated with the development of smart contracts for blockchains. As a result, we propose a pattern and its transformation rules for converting SLCML code to a choreography model and then implementing solidity smart contract code. For demonstration purposes, we deploy and implement our SLCML contract in Caterpillar, an open-source Blockchain-based BPM system that translates business processes modelled in BPMN into smart contracts written in Solidity language. As future work, we intend to create a tool-supported process for converting SLCML contract specifications into smart-contract code, such as Solidity, and to conduct more case studies with SLCML in blockchain research projects. We intend to develop a translator based on proposed transformation rules for the automatic conversion of SLCML instantiations into a larger set of blockchain-based languages.

References

1. Solidity — Solidity 0.7.1 documentation, https://docs.soliditylang.org/en/v0.7.1/
2. Aung, M.M., Chang, Y.S.: Traceability in a food supply chain: Safety and quality perspectives. Food Control **39**, 172–184 (2014). https://doi.org/https://doi.org/10.1016/j.foodcont.2013.11.007, https://www.sciencedirect.com/science/article/pii/S0956713513005811
3. Behnke, K., Janssen, M.: Boundary conditions for traceability in food supply chains using blockchain technology. International Journal of Information Management **52**, 101969 (2020). https://doi.org/https://doi.org/10.1016/j.ijinfomgt.2019.05.025, https://www.sciencedirect.com/science/article/pii/S0268401219303536
4. Boudjema, E.H., Verlan, S., Mokdad, L., Faure, C.: Vyper: Vulnerability detection in binary code. Security and Privacy **3**(2), e100 (2020). https://doi.org/https://doi.org/10.1002/spy2.100, https://onlinelibrary.wiley.com/doi/abs/10.1002/spy2.100
5. Brahem, A., Messai, N., Sam, Y., Bhiri, S., Devogele, T., Gaaloul, W.: Blockchain's fame reaches the execution of personalized touristic itineraries. In: 2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE). pp. 186–191 (2019). https://doi.org/10.1109/WETICE.2019.00047
6. Caro, M.P., Ali, M.S., Vecchio, M., Giaffreda, R.: Blockchain-based traceability in agri-food supply chain management: A practical implementation. In: 2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany). pp. 1–4 (2018). https://doi.org/10.1109/IOT-TUSCANY.2018.8373021
7. Casino, F., Kanakaris, V., Dasaklis, T.K., Moschuris, S., Rachaniotis, N.P.: Modeling food supply chain traceability based on blockchain technology. IFAC-PapersOnLine **52**(13), 2728–2733 (2019). https://doi.org/https://doi.org/10.1016/j.ifacol.2019.11.620, https://www.sciencedirect.com/science/article/pii/S2405896319316088, 9th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2019
8. Chen, T., Ding, K., Hao, S., Li, G., Qu, J.: Batch-based traceability for pork: A mobile solution with 2d barcode technology. Food Control **107**, 106770 (2020). https://doi.org/https://doi.org/10.1016/j.foodcont.2019.106770, https://www.sciencedirect.com/science/article/pii/S0956713519303597
9. Dannen, C.: Introducing Ethereum and solidity, vol. 318. Springer (2017)
10. Dwivedi, V., Deval, V., Dixit, A., Norta, A.: Formal-verification of smart-contract languages: A survey. In: Singh, M., Gupta, P., Tyagi, V., Flusser, J., Ören, T., Kashyap, R. (eds.) Advances in Computing and Data Sciences. pp. 738–747. Springer Singapore, Singapore (2019)
11. Dwivedi, V., Norta, A., Wulf, A., Leiding, B., Saxena, S., Udokwu, C.: A formal specification smart-contract language for legally binding decentralized autonomous organizations. IEEE Access **9**, 76069–76082 (2021). https://doi.org/10.1109/ACCESS.2021.3081926
12. Dwivedi, V., Pattanaik, V., Deval, V., Dixit, A., Norta, A., Draheim, D.: Legally enforceable smart-contract languages: A systematic literature review. ACM Comput. Surv. **54**(5) (Jun 2021). https://doi.org/10.1145/3453475, https://doi.org/10.1145/3453475
13. Efanov, D., Roschin, P.: The all-pervasiveness of the blockchain technology. Procedia Computer Science **123**, 116–121 (2018). https://doi.org/https://doi.org/10.1016/j.procs.2018.01.019, https://www.sciencedirect.com/science/article/pii/S1877050918300206, 8th Annual International Conference on Biologically Inspired Cognitive Architectures, BICA 2017 (Eighth Annual Meeting of the BICA Society), held August 1-6, 2017 in Moscow, Russia
14. Eshuis, R., Norta, A., Kopp, O., Pitkänen, E.: Service outsourcing with process views. IEEE Transactions on Services Computing **8**(1), 136–154 (2015). https://doi.org/10.1109/TSC.2013.51
15. Frantz, C.K., Nowostawski, M.: 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). pp. 210–215 (2016). https://doi.org/10.1109/FAS-W.2016.53
16. Frantz, C.K., Nowostawski, M.: 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). pp. 210–215. IEEE (2016)
17. Genestier, P., Zouarhi, S., Limeux, P., Excoffier, D., Prola, A., Sandon, S., Temerson, J.M.: Blockchain for consent management in the ehealth environment: A nugget for privacy and security challenges. Journal of the Interna-

tional Society for Telemedicine and eHealth **5**, (GKR);e24:(1–4) (Apr 2017), https://journals.ukzn.ac.za/index.php/JISfTeH/article/view/269

18. He, X., Qin, B., Zhu, Y., Chen, X., Liu, Y.: Spesc: A specification language for smart contracts. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). vol. 01, pp. 132–137 (2018). https://doi.org/10.1109/COMPSAC.2018.00025
19. He, X., Qin, B., Zhu, Y., Chen, X., Liu, Y.: Spesc: A specification language for smart contracts. In: 2018 IEEE 42nd Annual computer software and applications conference (COMPSAC). vol. 1, pp. 132–137. IEEE (2018)
20. Howson, P.: Building trust and equity in marine conservation and fisheries supply chain management with blockchain. Marine Policy **115**, 103873 (2020). https://doi.org/https://doi.org/10.1016/j.marpol.2020.103873, https://www.sciencedirect.com/science/article/pii/S0308597X19307067
21. López-Pintado, O., Dumas, M., García-Bañuelos, L., Weber, I.: Dynamic role binding in blockchain-based collaborative business processes. In: Giorgini, P., Weber, B. (eds.) Advanced Information Systems Engineering. pp. 399–414. Springer International Publishing, Cham (2019)
22. Miraz, M.H., Ali, M.: Applications of blockchain technology beyond cryptocurrency. Annals of Emerging Technologies in Computing **2**(1), 1–6 (Jan 2018). https://doi.org/10.33166/aetic.2018.01.001, http://dx.doi.org/10.33166/AETiC.2018.01.001
23. Moe, T.: Perspectives on traceability in food manufacture. Trends in Food Science & Technology **9**(5), 211–214 (1998). https://doi.org/https://doi.org/10.1016/S0924-2244(98)00037-5, https://www.sciencedirect.com/science/article/pii/S0924224498000375
24. Nakamoto, S., Bitcoin, A.: A peer-to-peer electronic cash system. Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf **4** (2008)
25. Norta, A., Ma, L., Duan, Y., Rull, A., Kõlvart, M., Taveter, K.: eContractual choreography-language properties towards cross-organizational business collaboration. Journal of Internet Services and Applications **6**(1) (Apr 2015). https://doi.org/10.1186/s13174-015-0023-7, https://doi.org/10.1186/s13174-015-0023-7
26. Orlenyslp: orlenyslp/caterpillar (2019), https://github.com/orlenyslp/Caterpillar/tree/master/v2.1/prototype
27. Porru, S., Pinna, A., Marchesi, M., Tonelli, R.: Blockchain-oriented software engineering: Challenges and new directions. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). pp. 169–171 (2017). https://doi.org/10.1109/ICSE-C.2017.142
28. Regnath, E., Steinhorst, S.: Smaconat: Smart contracts in natural language. In: 2018 Forum on Specification Design Languages (FDL). pp. 5–16 (2018). https://doi.org/10.1109/FDL.2018.8524068
29. Regnath, E., Steinhorst, S.: Smaconat: Smart contracts in natural language. In: 2018 Forum on Specification Design Languages (FDL). pp. 5–16 (2018). https://doi.org/10.1109/FDL.2018.8524068
30. von Rosing, M., White, S., Cummins, F., de Man, H.: Business process model and notation-bpmn. (2015)
31. Szabo, N.: Smart contracts. Unpublished manuscript (1994)
32. Wöhrer, M., Zdun, U.: Domain specific language for smart contract development. In: IEEE International Conference on Blockchain and Cryptocurrency (2020), http://eprints.cs.univie.ac.at/6341/

Appendix 5

V

V. K. Dwivedi and A. Norta. A legally relevant socio-technical language development for smart contracts. In *Proceedings - 2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems, FAS*W 2018*, pages 11–13. Institute of Electrical and Electronics Engineers Inc., 2018

2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems

A Legally Relevant Socio-Technical Language Development for Smart Contracts

Vimal Dwivedi
Department of Software Science
Tallinn University of Technology (of Aff.)
Tallinn, Estonia
vimal.dwivedi@ttu.ee

Supervisor: Prof. Alex Norta
Department of Software Science
Tallinn University of Technology
Tallinn, Estonia
alex.norta@ttu.ee

***Abstract*—Smart contracts play an advent role in automated business participation by rendering collaboration processes more time efficient, cost-effective and establishing more transparency. Smart contracts facilitate trust-less systems, without the need for intervention from third-party intermediaries. Existing smart-contract languages mainly focus on technical utility and do not take into consideration social and legally relevant issues, e.g., lack of semantics, ontological completeness, and so on. In this research, we address the gap by developing with rigorous means a smart contract's language that aims to be legally relevant, and that comprises socio-technical utility for cross-organizational business collaboration. The proposed language seeks to retain the strengths of the already existing languages of different generations while eluding their limitations. We aim to identify and implement abstract grammar patterns for a smart- contract language that has the expected application utility and verifiability. We evaluate the developed language based on automating industry-collaboration cases with our novel smart-contract language to test the suitability, utility and expressiveness.**

***Index Terms*—Ontological completeness, Suitability, Expressiveness, Socio-technical, ANTLR, Blockchain, Smart contracts**

I. Motivation

The blockchain is an incorruptible distributed ledger, a trust-less system that is duplicated across multiple networks [4]. Therefore, an application that could run previously through centralized medium only, now it can make possible without the trusted medium such as smart property, e-healthcare. With the emergence of blockchain technology, smart contracts have become popular because it can now operate in a decentralized fashion without the requirement of the trusted third-party. The smart contract is self-enforceable, self-executable, written in a program code, runs on a blockchain network [3]. Smart contract code executed itself when a set of pre-determined conditions met during the life-cycle of contract execution.

With respect to existing smart-contract languages, the most notable version such as Solidity and Rohlang are more recent adoption by industry. However, not only solidity but another existing smart-contract language does not have the social and legal semantics of business collaboration [7]. The social utility comprises the mechanism to maintain the transparency to all relevant stakeholders of business collaboration if conflicts arise [1]. The legal utility comprises the semantics of efficient breach of smart contracts as per law and economic [6].

Recently an experiment of the smart contract has been accomplished with the crowd-funding project named decentralized autonomous organization (DAO). It was hacked because of security flaws in smart-contract language, resulting in losses of money. This incident shows it is not sufficient to develop a contract with Turing-complete language such as solidity. Instead, it is essential to study suitability and expressiveness of the language [9]. Suitability means that smart-contract languages comprise the concept and properties to allow formulation of real-world contracts in the legally relevant way. Expressiveness implies the libraries of smart-contract languages have mathematical clarity that ensures the uniform enactment by several business process engines.

In [12], the proposed methodology uses solidity programming language to demonstrate the feasibility of untrusted business-process monitoring and execution in smart contract. However, the solidity does not take into account the ontological suitability and expressiveness because of Turing-complete nature. The core ontological concepts answer the conceptual questions of Who, Where and What about contracting parties. Without such an suitability and expressiveness, if a contract is valid and free from security issues, then verification of parties is not possible before the enactment. Therefore, the goal of this work is to develop a smart-contract programming language that incorporates socio-technical suitability and expressiveness for guiding business collaboration in a legally relevant way.

II. Objectives

This paper fills the gap in the current state of the art by posing the main research question how to develop a smart-contract language that has concept and properties for guiding business collaboration in a legally relevant way. From there we deduce several sub-questions. How to establish legal relevance in smart contracts that have socio-technical utility in smart contracts? How to design a language that combines the strengths of established languages of various generations while avoiding the weaknesses? How to identify and implement abstract grammar patterns for a smart-contract language that has the expected application utility and verifiability? From the first sub-question, we achieve the concept and properties

978-1-5386-5175-9/18/$31.00 ©2018 IEEE
DOI 10.1109/FAS-W.2018.00016

of business collaboration in legal relevance by developing an ontological structure of real-world contracts.

In order to aid the automation of contracting in business collaborations, it is required that working smart contracts could be produced by using the choreography language eSML [10] as a foundation. However, at the moment of writing, no solutions exist for mapping high-level choreography languages to smart-contract languages, such as the smart contracting lingua franca Solidity, while maintaining legal recognisability. Therefore, we need to deduce first sub-questions into several sub-questions. What are the requirement sets for an option to intent-fully not execute a contractual obligation, for the formation of contracts, and for party identification in smart contracts? What is the ontological structure of an option from legally enforceable requirements sets? What is the dynamic processing of ontological attributes?

From the first sub-question, we generate ontological structure and dynamic processing of business process agreements that work as input for designing smart-contract language. Therefore, we focus on second sub-question how to design a language that combines the strength of establishing various generation language while eluding their limitations. From the third sub-question, we achieve mathmatical clarity of the smart-contract language. Therefore, we focus on the semantics of context-free grammar to ensures uniform enactment by different process engines [8].

III. Methodology

The choice for research methodology for this thesis is the approach of the design science research methodology [2]. The authors of [2] described, "the design-science paradigm seeks to extend the boundaries of human and organisational capabilities by creating new and innovative artifacts." Hevner et al. [2] propose research principles, to conduct design science research (DSR), to create new and innovative artifacts, and for understanding, executing and evaluating information systems (IS) research. We focus on the following design-science research principles to generate results.

- **Design as an Artifact:** Our contribution is to produce artifacts in the form of constructs as a smart-contract language.
- **Problem Relevance:** The objective of this research is to develop smart-contract language that has socio-technical suitability and expressiveness for business collaboration.
- **Design Evaluation:** The utility, quality, and efficacy of a smart-contract language rigorously demonstrated via ANTLR tool.
- **Research Contributions:** Our first contribution is to develop an ontological structure of socio-technical and legal utilities of the business collaboration. Then follows the designing of language that combines of strengths of established languages of various generations while avoiding the weakness. Finally, we focus on the implementation of abstract grammar pattern for smart-contract language.
- **Research Rigor:** Existing tools from Truffle, will be used to emulate calls to a blockchain where these smart contracts are deployed, and write automated tests to ensure they work as expected.
- **Design as a Search Process:** Comparison of the results obtained from the artifact as a smart-contract language with existing languages, to reach desired goals.
- **Communication of Research:** We will present our contribution effectively both to technology-oriented as well as management-oriented audiences.
  A potential solution to the identified research goal is to use the eSML schema previously defined by Norta [10], to develop a context-free grammar. Therefore, We focus on existing tools such as ANTLR to help reach closer to the solution of our research goal. ANTLR has a consistent syntax for specifying lexers, parsers, and tree parsers [11]. We will evaluate the language based on automating industry-collaboration cases with our novel smart contract language to test suitability, utility and expressiveness.

IV. Research Plan

We plan the time-line of our research activities as per the design-science guidelines in the table 1. Till the moment, we have accomplished the activities of problem relevance, research questions and finalized the methodology.

Table 1: Time line for Ph.D. Research

Activity	2018 Jan-Dec	2019 Jan-Dec	2020 Jan-Dec	2021 Jan-Dec
Problem Relevance: Identifying, analyzing and categorizing the challenges of smart contracts languages	✓			
Research Contribution 1: Design the ontological structure of suitable semantics of business collaboration	✓	✓		
Research Contribution 2: Identify the suitable libraries for designing ontological based smart contract language		✓	✓	
Research Contribution 3: Identify and implement abstract grammar pattern for a smart contract language			✓	
Design as an artifact /Design evaluation / Design as a search process: Composition of the thesis and preliminary defense			✓	✓
Communication of Research: The defense of the thesis				✓

References

[1] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (sok). In *International Conference on Principles of Security and Trust*, pages 164–186. Springer, 2017.

[2] Martin Bichler. Design science in information systems research. *Wirtschaftsinformatik*, 48(2):133–135, 2006.

[3] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 2014.

[4] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.

[5] ConsenSys. *trufflesuite/truffle*, 2015 (accessed March 3, 2018). https://github.com/trufflesuite/truffle.

[6] Patrick Dahm. *The Efficient Breach of Smart Contracts*. http://learn.asialawnetwork.com/2018/02/22/efficient-breach-smart-contracts/.

[7] Mark Giancaspro. Is a smart contractreally a smart idea? insights from a legal perspective. *Computer Law & Security Review*, 33(6):825–835, 2017.

[8] Donald E Knuth. Semantics of context-free languages. *Mathematical systems theory*, 2(2):127–145, 1968.

[9] Alex Norta, Lixin Ma, Yucong Duan, Addi Rull, Merit Kõlvart, and Kuldar Taveter. econtractual choreography-language properties towards cross-organizational business collaboration. *Journal of Internet Services and Applications*, 6(1):8, 2015.

[10] Alexander Horst Norta. Exploring dynamic inter-organizational business process collaboration. *Dissertation Abstracts International*, 68(04), 2007.

[11] Terence Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

[12] Ingo Weber, Xiwei Xu, Régis Riveret, Guido Governatori, Alexander Ponomarev, and Jan Mendling. Untrusted business process monitoring and execution using blockchain. In *International Conference on Business Process Management*, pages 329–347. Springer, 2016.

Appendix 6

VI

V. Dwivedi, V. Deval, A. Dixit, and A. Norta. Formal-Verification of Smart-Contract Languages: A Survey. In *Advances in Computing and Data Sciences*, pages 738–747. Springer Singapore, 2019

Formal-Verification of Smart-Contract Languages: A Survey

Vimal Dwivedi( ), Vipin Deval, Abhishek Dixit, and Alex Norta

Department of Software Science, Tallinn University of Technology, Akadeemia tee 15A, 12816 Tallinn, Estonia
vimal.dwivedi@ttu.ee

Abstract. A blockchain is a peer-to-peer electronic ledger of transactions that may be publicly or privately distributed to all users. Apart from unique consensus mechanisms, their success is also obliged to smart contracts. Also, These programs let on distrusting parties to enter reconciliation that are executed autonomously. Although a number of studies focus on security of introducing new programming languages., However, there is no comprehensive survey on the smart-contract language in suitability and expressiveness concepts and properties that recognize the interaction between people in organizations and technology in workplaces. To fill this gap, we conduct a systematic analysis about smart-contract language properties that focus on e-contractual and pattern-based exploration. In particular, this paper gives smart-contract language taxonomy, introducing technical challenges of languages as well as recent solutions in tackling the challenges. Moreover, this paper also represents the future research direction in the introducing new smart-contract language.

Keywords: Ontological completeness · eSML · Socio-technical · Legal relevance · ANTLR · Block-chain · Smart contracts

1 Introduction

With traditional contract system, contract [14] refers to a liability that specifies legal action or liability that is required at the time of business collaboration as per the layout of terms and agreements, the obligation is formed. The contract includes agreements for binding parties together in terms and conditions so that they can collaborate businesses under a set of rules without any discrepancies. In traditional contract, every terms and condition which is written in the contract must be fulfilled in order to protect everyone's legal rights as communicated by an expert or lawyers. It overcomes the chances of risk, provides clarity of party expectation that is specified in contracts, enhances enforcement, limits flexibility. The conventional contract is a trust-based centralized system which requires intermediaries that leads to alleviated cost and is usually time-consuming.

With the emergence of blockchain technology, Smart contract [17] has become a necessity. It refers to a self-executing computer code, supervised by nodes. It

M. Singh et al. (Eds.): ICACDS 2019, CCIS 1046, pp. 738–747, 2019.
https://doi.org/10.1007/978-981-13-9942-8_68

runs on a distributed Ledger which thrives to achieve a trustless based system without the involvement of any intermediaries. It further accelerates the processing of the business processes and helps in attaining a higher accuracy rate. Another way to understand the smart contract is to compare the technology to a vending machine. Ordinarily, when you need water, you just need to drop a coin into the vending machine like a Ledger or escrow in another form and get water from the machine. Traditional contract system [14] works on every obligation in the exact same way but lacks the ability of self-execution as in the case of smart contract. The smart contract can applicable in numerous domains such as vehicle self-parking, real estate, healthcare, electronic voting because of its ability of self-execution and independence from the involvement of intermediaries.

With respect to current smart contract languages such as ethereum solidity language [6], Bitcoin scripting language and other existing procedural languages are facing vivid challenges such as adoption of issues relating to social meaning and legal relevance caused by incorrect arrangement between semantics and programmer intuition. It leads towards development of a new smart contract language that contains construct to deal with domain-specific aspects having social control [16] and artifact of law [7].

With smart contract ontology [11], we explore the eSourcing markup language (eSML) that specifies various aspects arising among collaborative business organizations due to the existence of process views for creating an agreement among the participating parties. In this study, we explore the socio-technical suitability and expressiveness for guiding business collaborations in a legally relevant way. Suitability means smart contract language encapsulates concept and properties and adoption in the construct of semantic which formulate into guiding business collaboration into the legally relevant way. With reference to smart contract language, we take into account solidity language of ethereum which is a key element in irreversibly changing the nature of the smart contracts. On the basis of smart contract ontology, we are focusing on the diversity of suitability and expressiveness among solidity, Rholang which is based on the property of syntactic language specification and ANTLR based language development. Solidity has played an essential role in the evolution of ontology for the smart contract. It has resulted in the efficient and effective management of concepts and properties that eSourcing Markup language embodies. However, Solidity does not support the pattern-based design, process awareness, and process matching. Inversely, these concepts are specified in smart contract ontology. The next generation language Rholang [13] captures these limitations. The utility required for handling business processes is one of the major aspects relating to collaborative business processes, but it is not handled properly with solidity as for Rholang. Unfortunately, the matching of processes, the process verification, and conclusion is not documented properly. ANTLR based language provides a powerful mechanism required to read, process, execute or translate the process according to its legal relevance. So in this study, we explore various suitability and expressiveness based languages that fulfil the limitations corresponding to the use of smart contracts i.e. of process awareness and verification requirement for establishing an automated business collaboration.

The hypothesis of the thesis states that the development of smart contract language must not be domain specific. It should adhere to socio-technical and legally relevance. This paper addresses the existing challenges in terms of socio-technical and legal relevance. In our thesis work, we will try to bridge the gap between the ideal situation and reality environment for the development of smart contract language.

2 State of the Art

Research in [14], gives an insight into the existence of a business contract. It provides knowledge regarding the use of contracts for facilitating communication between the participating entities. It further explains factors that lead to differences between the two most widely used types of contracts i.e. transactional and relational contracts and that the latter aids in its implementation without litigation or conflicts. The study emphasizes that contracts are focused more on strengthening business relationships rather than enforcing laws of the agreement.

A solution to the most fundamental problem in the implementation of Blockchain technology is provided in [18]. This naive technique addresses the issues related to trust in collaborative process enforcement using block-chain technology and its smart contracts to surpass the need of a centralized party. The proposed methodology comprises a three-step process with translator, blockchain infrastructure and triggers being major components.

A detailed introduction on bitcoin and its usage is illustrated in [10]. The subject deals with the evolution of bitcoin as a virtual currency, or decentralised digital currency coined by Satoshi Nakamoto in 2008 and its usage in public domain since early 2009. Bitcoin is used in an electronic payment business system which is based on cryptographic proof in lieu of trust and not managed by any financial authority (institution). Bitcoins are based on the pillars of transactions, proof of work, mining and digital wallet. It further provides a detailed expression on the working of bitcoin transactions, collection of transactional data into blocks, its peer to peer status, anonymity of users among others and formation of block-chain. The work further gives insight into the security issues and legal considerations in the use of bitcoins.

This paper [11] presents an esourcing ontology which is used as input for developing eSourcing markup language. Esourcing ontology comprises essentially concepts for the decentralized autonomous organization. Automating socio-technical business collaboration promises several benefits, including increases in efficiency, effectiveness, and quality, for developing a new generation of the so-called decentralized autonomous organization.

Norta et al. [11] presented the utility such as suitability and expressiveness as a gap in the cross-organizational business organization. In addition, Norta [12] presents esourcing markup language (eSML) which worked as choreography language for the automated cross-organization business process. Norta focuses on basis contractual elements for making the smart contract as a legal. Furthermore, a reduction of contractual elements that helps for improving difficulty level for legal issues such as obligation, artifacts of law. This is crucial when you are focusing on the cross-organizational business process in which less trusted participant involved [18].

Norta developed an esourcing ontology as a framework for building automated smart contract in a business collaboration [18] which would provide immutability and auditability. At a moment of writing contract in the collaborative business process, no mechanism exists for mapping high-level choreography language to smart contract language while maintaining legal recognisability.

Solidity [6] is a more popular language for smart contract development that runs on ethereum virtual machine. Butrin presents intention behind combining ethereum and EVM [3] for improving the concept of scripting, altcoins and on chain protocol that achieves consensus-based application that has scalability, feature completeness, interoperability and ease of development. The code is written in a low-level stack-based language and compiled into ethereum virtual machine in the byte code.

Despite the popularity, There are several reasons which make the implementation of smart contracts particularly prone to [security] errors in Ethereum [1]. The main reason is misalignment between semantics and intuition of the programmer. Solidity has various vulnerability such as the execution runs out of gas, call stack reaches its limit, the command throw is executed.

This paper [16] presents socio-technical utility for an autonomous business organization that enables flexible governance by providing organized structure in a way that has social meaning control of participants and high level of trust between parties. Additionally, highlights the compact contrast vision with existing approaches.

With the evolution of blockchain technology, computerized transaction protocol offering high reliable of trust, less transactional cost in which terms and condition are executed autonomously. This paper [7] expose the gap by considering potential issues for that smart contract considerable difficulty for adopting the current legal framework.

3 Problem Statement and Contributions

A study of the existing development scenario of smart contract language revealed that it has overlooked socio-technical perspectives during development. It means that it does not specify social control, social meaning, and utility for legal relevance. This paper exploits the address gap by investigating ideal and real world situation of existing business collaborations. Due to lack of semantics smart contract languages does not support cross-organizational business processes such as pattern-based design, process awareness, matching of the processes etc. Further, we examine transparency control in block-chain system in case of disputes among stakeholders due to lack of interdependence for the action. It exploits the address gap in existing smart contract programming language. Another major limitation is the difficulty to recover the losses that may arise if a faulty contract is embedded in the blockchain which may be due to misalignment between language library and intuition of the programmer [1].

Contract law for automated business collaboration is another research challenge that specifies assurance of terms and conditions. Therefore, we examine the address gap for legal relevance such as establishing capacity, contracting under a mistake, formation via technology and determine the conditions for the offer and acceptance of the contract. The essential part is establishing legal intent in 'follow-on contracting', the certainty of terms imbibed in the smart contract [7].

Generally, the smart contract does not include obligation duty which is a major issue pertaining to its legal relevance. To every mutually agreed upon the law in the smart contract, there must be an obligation duty so that we can overcome this above said issue. Along with that, we must have some screening procedure so that we can determine the age of the participating individual prior to his entry in block-chain transaction system so that we can establish legal intent for establishing capacity.

Our planned contribution includes the development of a language that has suitability and expressiveness for automated business collaborations in order to achieve higher efficiency, transparency among participants and automatic verification. This will help in fixing the address gap for the socio-technical relevance of the smart contract. We planned to enable contract for business collaborations by fixing the certainty of terms and condition and remedial issues which are revetment in smart contract block-chain.

This paper fills the gap by posing the research question how to develop a smart contract language that has the utility for guiding business collaboration in a legally relevant way? By posing this question, We examine the utilities for socio-technical such as appropriate semantics, participants control, process awareness, process matching that has legal relevance. After that, we deduce the main sub question into several sub question by fixing the sequence of order that specifies the proper layout to handle the address gap. To answer this question, a number of challenges (sub-question) need to be addressed.

The first sub-question pertains to establish legal relevance for a smart contract that has socio-technical utility. To justify this question, we focus on various grammar available for development and the requirements for business collaboration in a legally relevant way. Another subtask is the development of the ontology for autonomous legal business collaboration. And also, the targeted business process in autonomous business collaboration.

One of the key elements of legal relevance is an obligation. Obligation means, rights i.e. lawfully enforced rules that must have correlative duty between two parties. Therefore, the second sub-question highlights a process to design a language that combines the strengths of established languages of various generations while overcoming their limitations. To provide a solution to this question, we focus on strength of libraries of various generations of languages that make the language suitable and expressive.

As the abstract grammar pattern is a key element for the language development, the third research questions adhere to identification and implementation of abstract grammar patterns for a smart contract language that has the expected application utility and verifiability.

4 Research Methodology and Approach

Methodology is a sequence of methods that are used in the particular area of study that specifies how to do a task in a systematic manner. We study various methodology such as action design science [15], case study research [15], experimental research method [4]. We examine design science research [9] which is most suitable for information system design that evaluates sociotechnical artifacts. We use an ANTLR tool in this research for designing and evaluation smart contract language. A potential solution to the identified research goal would be to use the eSML schema previously defined by Norta, in order to develop a context-free grammar. The next step would be the use of existing tools such as ANTLR to help reach closer to the solution of our research goal. Use of ANTLR to develop such grammar comes with certain constraints such as the grammar produced using the tool has to be in an explicit format, as ANTLR notation.

In subsequent phases, ANTLR can then translate "recursive descent parsers from grammar rules .. which are exceptionally identical to the hand build rules by an adept programmer." By definition "recursive descent parsers are actually a collection of recursive methods, one method per rule derived. The term descent relates to the fact the starting point of parsing is the root i.e. the topmost node of a parse tree and it gradually proceeds towards the lowermost nodes i.e. the leaves as the rule gets more refined." as explained by Parr. The data structure used by ANTLR parsers are parse trees and this data structure is used to record "the way the parser perceives the complete structure of the input i.e. sentence and its constituent phrases." The next step is the implementation of a custom parse tree walker by applying the parsers generated using ANTLR tool. The parse tree walker so developed is aimed towards triggering callbacks on the identification of distinct tokens. The tokens so identified would then be transcribed into the solidity code parts with the aid of already acknowledged callbacks. The final phase would eventually be the development of smart contract after the parsing is exhaustively over.

The results so obtained can be validated from the fact that in our work we are using context-free grammar to define and explain the schema of the languages [8], which is the most widely used and accepted method of describing languages. Further, to verify and validate the working of obtained smart contracts, already-in-use tools from Truffle, will be referred to imitate calls to the blockchain, i.e. the smart contracts deployment site. Also, automated tests will be used to assure that the deviation between the actual and expected working on smart contracts is void. "Truffle is a development environment, testing framework and asset pipeline for Ethereum, aiming to make life of an Ethereum developer easier" [5] The research area, in general, is a prevailing research trend, as the underlying architecture i.e. blockchain and also, smart contract development technology is a naive concept and have been attracting smart programmers for evolution. Quoting Bartoletti et al. [2]: "In particular, the public and append-only ledger of transaction (the blockchain) and the decentralized consensus protocol that Bitcoin nodes use to extend it, have revived Nick Szabo's idea of smart contracts i.e. programs whose correct execution is automatically enforced without relying on a trusted authority [17]".

5 Preliminary or Intermediate Results

The research problem addressed by us can be considered as an intermediate result for examining sociotechnical utility [16] and legal relevance [7]associated with smart contracts. Ideally, the start contract must possess process awareness to avoid situation of security breach and transparency to all the participating parties for smooth execution in case a conflict arises. Along with this, verification of the correctness of the smart contract being developed is provided to improve its effectiveness at execution time and efficiency of cross organisational business.

The address gap extracted focuses on socio technical and legal relevance elements for the development of a smart contract language. It also includes issues relating to lack of control for participants, dearth of social meaning and understanding of business process in cross organisational infrastructure. Also, we examined and identified various impediments for adopting existing legal framework of the contract.

These findings set the vision of our research work, to improve the language development that combines the social element of business collaboration in a legal framework.

6 Discussion

This section discusses the study results and answers the research questions that we defined in Sect. 3.

Ethereum is one of the most suitable platforms to propose the conception of smart contracts in the blockchain. It boasts the turing completeness of its smart contract platform. The language used i.e. Solidity is good enough to make it Turing complete but it does lack in the flexibility which is provided by the languages used today. Wanchain's distributed ledger relies on the strengths of Ethereum, and any Ethereum DApp will run on Wanchain without any code alteration. To enhance these applications, Wanchain uses solidity that offers a number of APIs designed to expand cross-chain capabilities and improve privacy protection.

The overreaching functional goal of Æternity blockchain smart contracts is to be able to runs code on the chain. That is, code execution that is verifiable by a miner and which can alter the state of the chain. For efficient contract execution Æternity provides a very high level language for blindingly fast execution of simple contracts. For more advanced contracts the Sophia language is used and that is compiled to a virtual machine tailored for execution of the contracts. This machine is a high level machine with instructions for operating on the chain and on Sophia data structures without any need to do explicit stack and memory management.

Zen blockchain uses a Total language to express smart contracts rather than depending on evaluation model which tracks gas in order to ensure the totality. Total languages are capable of expressing arbitrary logic like recursion and loops, and this is also the case for Zen Protocol. Zen's smart contracting language is 'Dependently Typed', i.e every expression has a type that depends on both the expressions and the types. Dependent type systems are expressive to use them for the purpose of the formal Verification. Such types can express arbitrary properties of expressions. Zen Protocol's smart contract takes dependently typed source code, that must express the resource consumption. Zen Protocol is very limited by the time taken to run the smart contracts, and therefore, must be able to process transactions involving smart contracts faster—smart contracts in Zen are not only faster to run, but can be executed most of the time in parallel.

Counterparty blockchain relies on Bitcoin for its consensus. But it also supports ethereum smart contracts. It uses solidity or serpent to write smart contract code and compile it to a more compact form (bytecode). Serpent is a language for smart contract development based on Python language. Python is arguably one of the best language for novice programmers, and a very productive language for experienced developers. Serpent, originally developed for Ethereum, is currently being used in complex enterprise projects.

RChain is a project that focuses on scalability by using a multi-threaded blockchain with its own smart contract language. Smart contracts employ a number of industry-leading functions such as meta-programming, reactive data streams, pattern matching. As a result, RChain contracts have programmability that can be used on RChain nodes. Rholang is a "process-oriented" i.e. computations being done by message passing. Messages are passed via "channels", that are like message queues.

Qtum is an Ethereum-based smart contracts system that run on top of a Bitcoin-based blockchain. It uses a modified version Blackcoin's Proof of Stake (PoS) implementation for consensus. Qtum has added the custom adaptation layer that maps the Ethereum account balances to sets of Bitcoin Unspent Transaction Outputs (UTXOs). Qtum is planning to extend their smart contracts offering to include a x86 virtual machine that will enable the smart contracts development in languages such as Java, C++, and Haskell. However, leverage with existing tooling, it doesn't specifically address the security issues inherent to Solidity's design.

7 Conclusions

Presently solidity language is being used to develop smart contracts in blockchain systems. It has some limitations associated with it such as pattern-based design, process awareness, matching of the process. In this study, we develop artifacts that incorporate socio-technical utility which enables to automate cross organizational business collaboration adhering to the legal relevance. Further, we develop a language for smart contracts that can preserve autonomy and transparency among participants in peer to peer decentralized infrastructure.

In our work, we will provide a language incorporating the strength of existing languages while overcoming their limitation. Future, we will refine the language features along with its testing in a realistic environment.

Acknowledgement. This Ph.D. research is partially supported by Quantum Foundation, Singapore under the supervision of Prof. Alex Norta, Associate Professor, Department of Software Science, Tallinn University of Technology, Tallinn, Estonia. Email: `alex.norta.phd@ieee.org` We thank our Prof. Alex Norta who provided insight and expertise that greatly assisted the research.

References

1. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on ethereum smart contracts (SoK). In: Maffei, M., Ryan, M. (eds.) POST 2017. LNCS, vol. 10204, pp. 164–186. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54455-6_8
2. Bartoletti, M., Pompianu, L.: An empirical analysis of smart contracts: platforms, applications, and design patterns. In: Brenner, M., et al. (eds.) FC 2017. LNCS, vol. 10323, pp. 494–509. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70278-0_31
3. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. White paper (2014)
4. Christensen, L.B.: Experimental Methodology. Allyn & Bacon, Boston (2004)
5. ConsenSys: trufflesuite/truffle (2015). https://github.com/trufflesuite/truffle. Accessed 3 Mar 2018
6. Ethereum: ethereum/solidity (2015). https://github.com/ethereum/solidity. Accessed 3 Mar 2018
7. Giancaspro, M.: Is a 'smart contract' really a smart idea? Insights from a legal perspective. Comput. Law Secur. Rev. **33**(6), 825–835 (2017)
8. Knuth, D.E.: Semantics of context-free languages. Math. Syst. Theory **2**(2), 127–145 (1968)
9. March, S.T., Storey, V.C.: Design science in the information systems discipline: an introduction to the special issue on design science research. MIS Q. **32**, 725–730 (2008)
10. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
11. Norta, A., Ma, L., Duan, Y., Rull, A., Kõlvart, M., Taveter, K.: eContractual choreography-language properties towards cross-organizational business collaboration. J. Internet Serv. Appl. **6**(1), 8 (2015)
12. Norta, A.H.: Exploring dynamic inter-organizational business process collaboration. Dissertation Abstracts International, 68(04) (2007)
13. RChain: RChain/rholang. https://steemit.com/smart/@alexbafana/smart-contract-languages-comparison
14. Roxenhall, T., Ghauri, P.: Use of the written contract in long-lasting business relationships. Ind. Mark. Manag. **33**(3), 261–268 (2004)
15. Sein, M.K., Henfridsson, O., Purao, S., Rossi, M., Lindgren, R.: Action design research. MIS Q. **35**, 37–56 (2011)
16. Singh, M.P., Chopra, A.K.: Violable contracts and governance for blockchain applications. arXiv preprint arXiv:1801.02672 (2018)

17. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997)
18. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 329–347. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_19

Curriculum Vitae

1. Personal data

Name	Vimal Kumar Dwivedi
Date and place of birth	17th August, Allahabad, India
Nationality	India

2. Contact information

Address	Tallinn University of Technology, School of Information Technologies, Department of Software Science, Ehitajate tee 5, 19086 Tallinn, Estonia
E-mail	vimal.dwivedi@ttu.ee

3. Education

2017–...	Tallinn University of Technology, School of Information Technologies, Computer science, PhD studies
2013–2015	Guru Gobind Singh Indraprastha University, Information Technology, Master of technology *cum laude*
2008–2012	Dr. A.P.J. Abdul Kalam Technical University, Information Technology, Bachelor of technology

4. Language competence

Hindi	native
English	fluent

5. Professional employment

2021– ...	University of Tartu, Tartu, Junior Lecturer
2015–2017	Krishna engineering College, India, Assistant Professor

6. Computer skills

- Operating systems: Windows
- Programming languages: Python, C, Sql, XML and Java

7. Honours and awards

- 2021, Best paper award for presenting a paper "Auto-Generation of Smart Contracts from Domain-Specific XML-Based Language" in FICTA-2021, Springer

8. Defended theses

- 2015, Offline Signature verification through wavelet feature extraction and SVM classifier, M.Tech, supervisor Assoc. Prof. Tushar Patanaik, Guru Gobind Singh Indraprastha University, India.

9. Field of research

- Blockchain Systems
- Requirement Engineering
- Machine Learning
- Supply Chain Analytics

Elulookirjeldus

1. Isikuandmed

Nimi	Vimal Kumar Dwivedi
Sünniaeg ja -koht	17th August, Allahabad, India
Kodakondsus	India

2. Kontaktandmed

Aadress	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, Ehitajate tee 5, 19086 Tallinn, Estonia
E-post	vimal.dwivedi@ttu.ee

3. Haridus

2017–. . .	Tallinna Tehnikaülikool, Infotehnoloogia teaduskond, arvutiteadus, doktoriõpe
2013–2015	Guru Gobind Singh Indraprastha ülikool, Infotehnoloogia, Tehnoloogia magister *cum laude*
2008–2012	Dr A.P.J. Abdul Kalami tehnikaülikool, Infotehnoloogia, tehnoloogia bakalaureus

4. Keelteoskus

hindi keel	emakeel
inglise keel	kõrgtase

5. Teenistuskäik

2021– . . .	Tartu Ülikool, Tartu, noorem õppejõud
2015–2017	Krishna insenerikolledž, India, dotsent

6. Arvutioskused

- Operatsioonisüsteemid: Windows
- Programmeerimiskeeled: Python, C, Sql, XML and Java

7. Autasud

- 2021, Parima artikli auhind artikli "Nutikate lepingute automaatne genereerimine domeenispetsiifilisest XML-põhisest keelest" eest konverentsil FICTA-2021 (Springer)

8. Kaitstud lõputööd

- 2015, Võrguühenduseta allkirja kontrollimine lainete funktsioonide eraldamise ja SVM-klassifikaatori kaudu, M.Tech, juhendaja Assoc. Prof. Tushar Patanaik, Guru Gobind Singh Indraprastha ülikool, India.

9. Teadustöö põhisuunad

- Plokiahela süsteemid
- Nõuete kavandamine
- Masinaõpe
- Tarneahela analüüs

ISSN 2585-6901 (PDF)
ISBN 978-9949-83-816-5 (PDF)