IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

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, Akadeemia tee 15A, Tallinn, Estonia (e-mail: vimal.dwivedi@taltech.ee, alex.norta@taltech.ee, chibuzor.udokwu@taltech.ee)
[2]SRH Hochschule, Berlin, Germany (e-mail: alexander.wulf@srh-hochschule-berlin.de)
[3]TU Clausthal, Adolph-Roemer-Straße 2A, 38678, Clausthal, Germany (e-mail: benjamin.leiding@tu-clausthal.de)
[4]Galgotias College of engineering and technology, Greater Noida, India (e-mail:sandeep.research29@gmail.com)

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

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 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, numerous collaborative parties are committed from production to delivery, and the integration of processes of each involved party requires widespread information interchange [5]. The lack of consistent information exchange poses a collaboration challenge for inter-organizational business processes [6]. Blockchain technology controls the execution of inter-organizational business processes through smart contracts and enables decentralized autonomous organizations (DAO) [7]. 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 [8]. The recent DAOs (such as The DAO) are controlled by the software community, seeking to re-implement traditional decision-making rules through blockchain technology [8]. 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 [9]. The concept of a smart contract is first time introduced in seminal work [10] 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 [11]. 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 [12]. 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.

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

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 [13]. 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 [14]. 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 [15], enactment [16], 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 [17]. 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 [18]. Furthermore, smart objects such as belief-desire-intention (BDI) agents can be combined with smart contracts to collaborate as self-aware DAOs [19].

We discover that several workarounds, for example, SPECS [20], Symboleo [21], SmaCoNat [22], 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 specification of a smart contract (rather than its implementation) for the purpose of 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, which prepares the reader to comprehend the subsequent sections. In Section III, we represent the formally-verified

[2]shorturl.at/gxFKT
[3]shorturl.at/uBHR6

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 [23], toll systems [24], etc. The significant use-case of blockchain is tracking and monitoring the vehicle parts in the automotive supply-chain [25]. 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) [26]. 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 [15]. 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) [27] in subsets of the internal in-house process [28]. 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) [29] 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 [15], 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 [15]. Phase 1 entails the extraction of proto-contracts, while Phase 2 is used for the consensual establishment of smart contracts. According to [30], 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 [15], 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 [15]. Phase 1 entails the extraction of proto-contracts, while Phase 2 is used for the consensual establishment of smart contracts. According to [30], 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 [32] 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 [33]. 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 can not 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 conceptual layers of the service providers are mapped into smart-contract blockchain systems on the respective internal

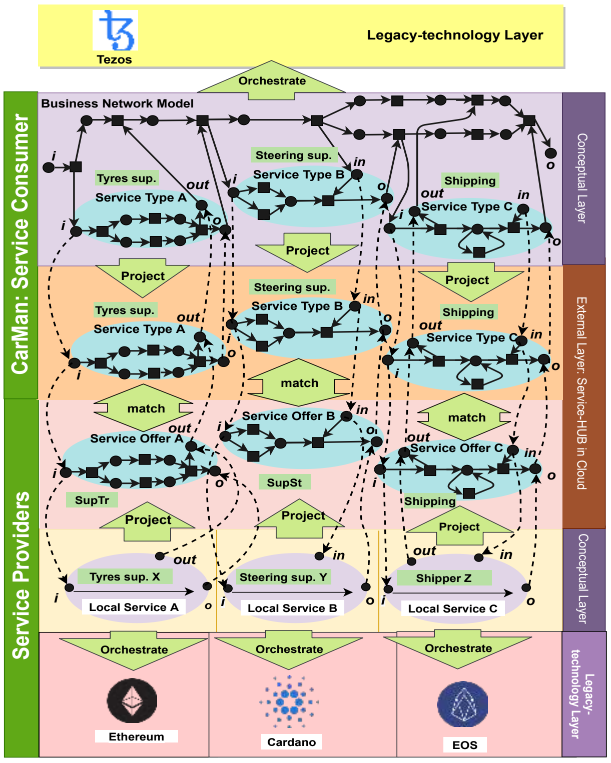

FIGURE 1. DAO-collaborative automotive supply-chain [31].

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 [34]. 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 can not 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 [35]. 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 [36]. 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 [37], and is used to overcome the conceptual inconsistencies in the blockchain domain [38]. 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 [39] for developing the SCL ontology, which is an open-source ontology editor and comprises a VOWL [40] 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 [41]. Next, we use the Liquid studio tool for mapping the ontological concepts and properties into XML schema. Liquid studio[4] provides an advanced toolkit for XML and JSON development along with the data mapping and transformation tools (such as XSLT-, XQuery editor etc.), comprising the graphical XML schema editor for visualizing, authoring and navigating complex XML schema. The former provides an interactive logical view of the XML schema, enabling intuitive editing while retaining the ability 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 [29]. In our previous work, the eSourcing framework is defined to specify and verify harmonized B2B process collaborations [42]. Based on the concept of eSourcing, the eSourcing ontology [29] 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. 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 [29] for further information about the collaboration model.

Since contracts can be of different types, the realm and range of each type differ vastly. Expressing the entire spectrum of contracts in a single ontology is difficult as the latter is too large and diverse in nature to be of practical use. A multi-tiered contract ontology that progressively moves from abstract to specific meta data definitions to stratifications is proposed to capture the full range of business-related contracts within a unified model. 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. Each contract-type inherits every fundamental characteristic of the upper-layer and then specializes in the particular knowledge specific to the contract domain, as presented in Figure 4.

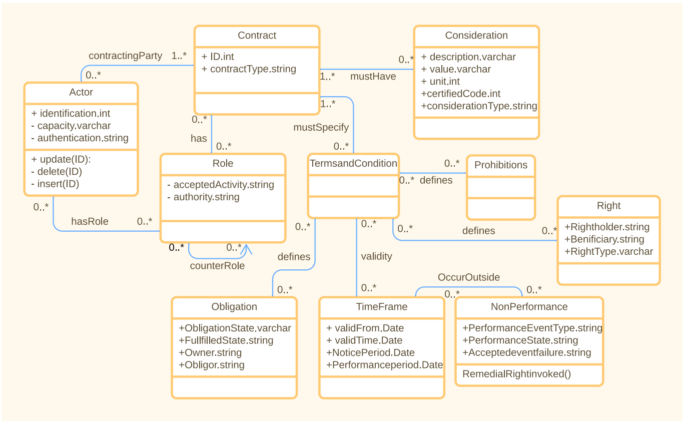

FIGURE 2. Outline for the upper-level smart-contract ontology.

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-case scenario. 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 commitment to perform some act, or to perform certain actions, e.g., supplies 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 [43]. If a party is unable

[4]Liquid Studio | Home

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. A consideration must be valuable (at least to the parties) and must be exchanged for performance. 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.

[5]DAO | Legal status
[6]Consideration | Legal Definition

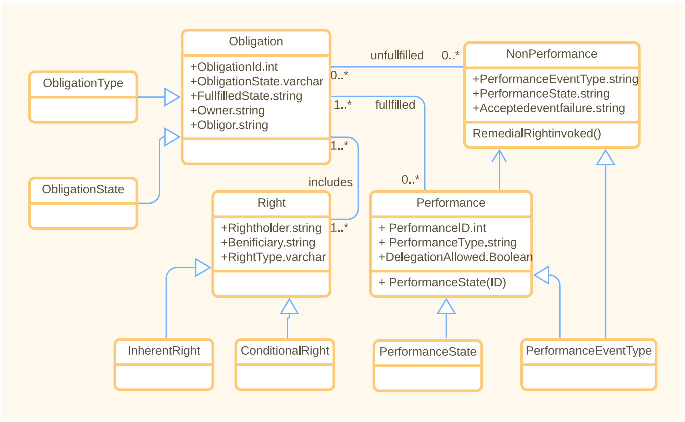

FIGURE 3. Rights and obligations.

B. SPECIFIC DOMAIN LAYER

The contract statements are informative, declarative, or performative, as discussed in [44], [45]. 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. For example, the CarMan obligation to pay is reciprocal to the SupTr obligation to deliver and vice versa. The CarMan obligation to pay is also a primary obligation of the former. A conditional obligation does not need to be triggered in the ordinary course of events. Most of the remedial rights and obligations fall into this category. For example, if CarMan

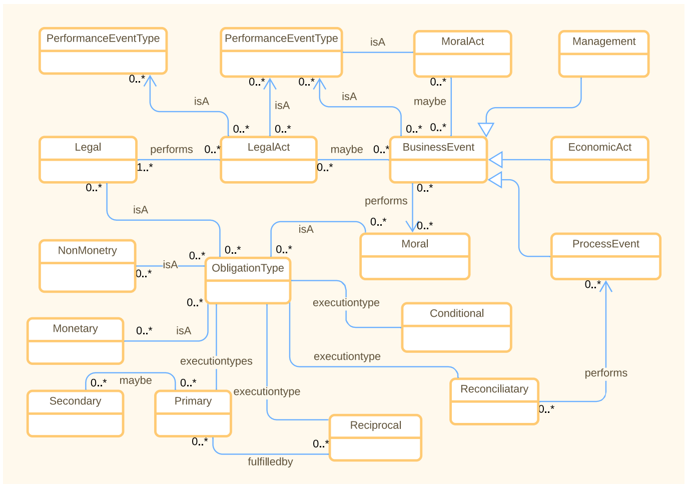

FIGURE 4. Specific domain layer.

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, 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.

[7]International chamber of commerce | Home

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 [46], 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 [29]. 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

[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

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.

```
<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.

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 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.

```
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.

`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 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.

```
<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.

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.

```
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 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.

```
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" />
6             <xs:element name="moral" type="xs:string" minOccurs="0" />
7             <xs:element name="Primary" type="xs:string" minOccurs="0" />
8             <xs:element name="Secondary" type="xs:string" minOccurs="0" />
9             <xs:element name="Conditional" type="xs:string" minOccurs="0" />
10            <xs:element name="reciprocal" type="xs:string" minOccurs="0" />
11            <xs:element name="reconciliatory" type="business_event_types" minOccurs="0" />
12        </xs:sequence>
13    </xs:complexType>
```

Listing 5. Schema of the type of obligation.

V. FEASIBILITY EVALUATION

For our automotive running case, we briefly discuss the SLCML code examples based on the presented SLCML schema in the previous section. Listing 6 shows an example of defining a legally-binding contract that has a unique ID and can not 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.

```
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.

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 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.

```
1 <obligation_rule tag_name ="paying_invoices" rule_id ="0001" changeable ="false" monetary = "true">
2     <state> enabled </state>
3    <parties>
4       <beneficiary> SupTr (31 x7 ) </beneficiary>
5       <beneficiary > SupSt (31 x7 ) </beneficiary>
6       <obligor> CarMan (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>
13      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.

The code extract of Listing 8 comprises intersecting provisions with the obligation. The rights and obligations are related, which means that if one party poses its rights, the corresponding party is obliged to adhere. 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.

```
<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.

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 enabled for immediate enactment, and the parties are defined similarly as in Listing 7. The right-type is set to 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 [47]. 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 [48]. Caterpillar is an open-source Blockchain-based BPM system that converts business processes modelled in BPMN into smart contracts written in Solidity language. Still, we do not discuss the transformation rules because they are beyond the scope of the paper. We only discuss the Solidity code presented

[10]Sagewise | Dispute resolution

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, 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 notifyObligationBreach 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.

```
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.

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 [20], 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 [21], 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 [49], 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 [22], 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 on the readability and safety of smart contracts and does not express the collaborative contractual suitability and completeness of the domain. In [50], 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

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]

from addressing the particular complexity of blockchains. CML allows blockchain developers to focus on the business process rather than the syntax details of each blockchain platform. The focus and scope of CML is completely different from our research. Attributes Deontic AIm Conditions (ADICO) [51] is a DSL developed in Scala that converts domain-specific constructs of smart contracts to simpler concepts. In [52], 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 [53] 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 [54], 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] Xiongfeng Pan, Xianyou Pan, Malin Song, Bowei Ai, and Yang Ming. Blockchain technology and enterprise operational capabilities: An empirical test. International Journal of Information Management, 52:101946, 2020.
[2] Paul Vigna and Michael J. Casey. The Age of Cryptocurrency: How Bitcoin and Digital Money Are Challenging the Global Economic Order. St. Martin's Press, Inc., USA, 2015.
[3] Christian Catalini and Joshua S. Gans. Some simple economics of the blockchain. Communications of the ACM, 63(7):80–90, June 2020.
[4] Yan Chen and Cristiano Bellavitis. Blockchain disruption and decentralized finance: The rise of decentralized business models. Journal of Business Venturing Insights, 13:e00151, 2020.
[5] Claudio Di Ciccio, Alessio Cecconi, Marlon Dumas, Luciano García-Bañuelos, Orlenys López-Pintado, Qinghua Lu, Jan Mendling, Alexander Ponomarev, An Binh Tran, and Ingo Weber. Blockchain support for collaborative business processes. Informatik Spektrum, 42(3):182–190, May 2019.
[6] Rik Eshuis, Alex Norta, and Raoul Roulaux. Evolving process views. Information and Software Technology, 80:20 – 35, 2016.
[7] Alex Norta. Designing a smart-contract application layer for transacting decentralized autonomous organizations. In Mayank Singh, P.K. Gupta, Vipin Tyagi, Arun Sharma, Tuncer Ören, and William Grosky, editors, Advances in Computing and Data Sciences, pages 595–604, Singapore, 2017. Springer Singapore.

[8] Madhusudan Singh and Shiho Kim. Chapter four - blockchain technology for decentralized autonomous organizations. In Shiho Kim, Ganesh Chandra Deka, and Peng Zhang, editors, Role of Blockchain Technology in IoT Applications, volume 115 of Advances in Computers, pages 115 – 140. Elsevier, 2019.

[9] N. Diallo, W. Shi, L. Xu, Z. Gao, L. Chen, Y. Lu, N. Shah, L. Carranco, T. Le, A. B. Surez, and G. Turner. egov-dao: a better government using blockchain based decentralized autonomous organization. In 2018 International Conference on eDemocracy eGovernment (ICEDEG), pages 166–171, 2018.

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

[11] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, Ingo Weber, and Alexander Ponomarev. Caterpillar: A business process execution engine on the ethereum blockchain. Software: Practice and Experience, 49(7):1162–1193, 2019.

[12] Christoph Jentzsch. Decentralized autonomous organization to automate governance. White paper, November, 2016.

[13] Nanjangud C. Narendra, Alex Norta, Msury Mahunnah, Lixin Ma, and Fabrizio Maria Maggi. Sound conflict management and resolution for virtual-enterprise collaborations. Service Oriented Computing and Applications, 10(3):233–251, Sep 2016.

[14] L. Sterling and K. Taveter. The art of agent-oriented modeling. 2009.

[15] Alex Norta. Creation of smart-contracting collaborations for decentralized autonomous organizations. In Raimundas Matulevičius and Marlon Dumas, editors, Perspectives in Business Informatics Research, pages 3–17, Cham, 2015. Springer International Publishing.

[16] Alex Norta. Establishing distributed governance infrastructures for enacting cross-organization collaborations. In Alex Norta, Walid Gaaloul, G. R. Gangadharan, and Hoa Khanh Dam, editors, Service-Oriented Computing – ICSOC 2015 Workshops, pages 24–35, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[17] Alex Norta, Anis Ben Othman, and Kuldar 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.

[18] Chibuzor Udokwu and Alex Norta. Deriving and formalizing requirements of decentralized applications for inter-organizational collaborations on blockchain. Arabian Journal for Science and Engineering, Mar 2021.

[19] Alex Norta. Self-aware smart contracts with legal relevance. In 2018 International Joint Conference on Neural Networks (IJCNN), pages 1–8, 2018.

[20] 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.

[21] 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.

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

[23] Fabian Knirsch, Andreas Unterweger, and Dominik Engel. Privacy-preserving blockchain-based electric vehicle charging with dynamic tariff decisions. Computer Science - Research and Development, 33(1-2):71–79, September 2017.

[24] 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.

[25] Shuchih Ernest Chang, Yi-Chian Chen, and Ming-Fang 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.

[26] 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.

[27] 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.

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

[29] 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), April 2015.

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

[31] Nanjangud C. Narendra, Alex Norta, Msury Mahunnah, Lixin Ma, and Fabrizio Maria Maggi. Sound conflict management and resolution for virtual-enterprise collaborations. Service Oriented Computing and Applications, 10(3):233–251, October 2015.

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

[33] Jian-Hong Lin, Kevin Primicerio, Tiziano Squartini, Christian Decker, and Claudio J Tessone. Lightning network: a second path towards centralisation of the bitcoin economy. New Journal of Physics, 22(8):083022, aug 2020.

[34] Yi-Hui Chen, Shih-Hsin Chen, and Iuon-Chang Lin. Blockchain based smart contract for bidding system. In 2018 IEEE International Conference on Applied System Invention (ICASI), pages 208–211, 2018.

[35] Xiang Hong 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.

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

[37] Alexander Maedche and Steffen Staab. Ontology learning for the semantic web. IEEE Intelligent systems, 16(2):72–79, 2001.

[38] Joost de Kruijff and Hans Weigand. Understanding the blockchain using enterprise ontology. In Eric Dubois and Klaus Pohl, editors, Advanced Information Systems Engineering, pages 29–43, Cham, 2017. Springer International Publishing.

[39] Mark A Musen et al. The protégé project: a look back and a look forward. AI matters, 1(4):4, 2015.

[40] Steffen Lohmann, Stefan Negru, and David Bold. The protégévowl plugin: Ontology visualization for everyone. In Valentina Presutti, Eva Blomqvist, Raphael Troncy, Harald Sack, Ioannis Papadakis, and Anna Tordai, editors, The Semantic Web: ESWC 2014 Satellite Events, pages 395–400, Cham, 2014. Springer International Publishing.

[41] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. Hermit: an owl 2 reasoner. Journal of Automated Reasoning, 53(3):245–269, 2014.

[42] Alex Norta and Rik Eshuis. Specification and verification of harmonized business-process collaborations. Information Systems Frontiers, 12(4):457–479, April 2009.

[43] Guido Governatori, Florian Idelberger, Zoran Milosevic, Regis Riveret, Giovanni Sartor, and Xiwei Xu. On legal contracts, imperative and declarative smart contracts, and blockchain systems. Artificial Intelligence and Law, 26(4):377–409, Dec 2018.

[44] Ronald M Lee and Sandra Donaldson Dewitz. Facilitating international contracting: AI extensions to edi. International Information Systems, 1(1):94–123, 1992.

[45] 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.

[46] I Horrocks, B Motik, and Z Wang. The hermit owl reasoner. volume 858, pages 245–269. CEUR Workshop Proceedings, 2012.

[47] Amy Schmitz and Colin Rule. Online dispute resolution for smart contracts. Journal of Dispute Resolution, 2019(2):103–126, 2019.

[48] Orlenyslp. orlenyslp/caterpillar, 2019.

[49] Maximilian Wöhrer and Uwe Zdun. Domain specific language for smart contract development. In IEEE International Conference on Blockchain and Cryptocurrency, 2020.

[50] Mohammad Hamdaqa, Lucas Alberto Pineda Metz, and Ilham 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.

[51] 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.
[52] Michele Soavi, Nicola Zeni, John Mylopoulos, and Luisa Mich. Contract – from legal contracts to formal specifications: Preliminary results. In Jānis Grabis and Dominik Bork, editors, The Practice of Enterprise Modeling, pages 124–137, Cham, 2020. Springer International Publishing.
[53] Orlenys López-Pintado, Marlon Dumas, Luciano García-Bañuelos, and Ingo Weber. Dynamic role binding in blockchain-based collaborative business processes. In Paolo Giorgini and Barbara Weber, editors, Advanced Information Systems Engineering, pages 399–414, Cham, 2019. Springer International Publishing.
[54] Andrea Pinna and Simona Ibba. A blockchain-based decentralized system for proper handling of temporary employment contracts. In Advances in Intelligent Systems and Computing, pages 1231–1243. Springer International Publishing, November 2018.

IX. 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".

VIMAL DWIVEDI is a Ph.D. student and early stage researcher of the blockchain technology group at Tallinn University of Technology. He received his masters degree in Information technology from Indraprashta university, Delhi. He has worked as assistant professor of Information Technology in India. Vimal has (co-)authored 4 publications in conference proceedings. His research interests are semantics and ontology development, and legally-relevant smart contract languages development for blockchains.

ALEX NORTA is currently a principal investigator at the Blockchain Technology Group and a research member at the Faculty of Software Science/TalTech in Tallinn/Estonia and was earlier a researcher at the Oulu University Secure-Programming Group (OUSPG) after having been a postdoctoral researcher at the University of Helsinki, Finland. He received his M.Sc. degree (2001) from the Johannes Kepler University of Linz, Austria, and his Ph.D. degree (2007) from the Eindhoven University of Technology, The Netherlands. 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. 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, e-governance. 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.

ALEXANDER J. WULF is a professor of business law at the Berlin School of Management. His research interests are the 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 his B.Sc. degree in computer science in 2015 from the University of Rostock, Germany. Subsequently, he received the M.Sc. degree in Internet Technologies and Information Systems in 2017 as well as the Ph.D. degree in computer science in 2020 from the University of Goettingen, Germany. He is currently a Post Doctoral Research Fellow at the Clausthal University of Technology. His research interests include the machine-to-everything Economy (M2X Economy), the Circular Economy, distributed systems, and digital identities.

SANDEEP SAXENA , PhD from NIT Durgapur, west Bengal andworking as Associate Professor in a reputed engineering institute Galgotias College of Engineering & Technology, Greater Noida. I had completed my B.Tech in CSE from UPTU Lucknow and MS in Information Security from IIIT Allahabad. I have more than 12 Years, Teaching Experience. I had performed the role of a key member in 6 International Conferences as Organizing Secretary/Organizing Chair/Session Chair. I had written 3 technical books for UP Technical University, Lucknow, and published multiple research papers in reputed international journals and conferences. I had published 8 international Conferences and 2 SCIE, 9 Patent published, 2 Scopus, and 6 other published in International journals. I am also participating in multiple professional societies like IEEE (Senior Member), IAASSE (Senior Member), CSI, and CRSI.

CHIBUZOR UDOKWU is an external Ph.D. student of TalTech in the blockchain technology group. He received his masters degree in Software Science, TalTech. He has consulted and help in designing several blockchain applications for different startups. Chibuzor has published and co-authored several scientific papers in the blockchain space. His research interests are semantics and ontology development, design and development of blockchain systems, blockchain use-cases and applications in organizations.

• • •