Towards an Automated DEMO Action Model Implementation using Blockchain Smart Contracts

Marta Aparício[1,2], Sérgio Guerreiro[1,2] and Pedro Sousa[1,2,3]
[1]*Instituto Superior Técnico, University of Lisbon, Av. Rovisco Pais 1, 1049-001 Lisbon, Portugal*
[2]*INESC-ID, Rua Alves Redol 9, 1000-029 Lisbon, Portugal*
[3]*Link Consulting SA, Av. Duque de Ávila 23, 1000-138 Lisbon, Portugal*

Keywords: Automatic Generation, Blockchain, DEMO, DEMO Action Model, Smart Contract.

Abstract: Blockchain (BC) is a technology that introduces a decentralized, replicated, autonomous, and secure databases. A Smart Contract (SC) is a transaction embedded in BC that contains executable code and its internal storage, offering immutable execution and record keeping. A SC has enormous potential in automating traditional paper contracts and encoding contract logic into program code. Therefore, replacing the role of a central authority and reducing the time and money spent on the enforcement of such contracts. This paper intends to determine the sufficiency or insufficiency of ontology to support the automatic generation of SCs code from text, in particular, DEMO Action Model. A new way to capture the SC in a user-friendly way could be proposed. With this, it is intended to eliminate the errors associated with programming since the SC code is automatically generated from models.

1 INTRODUCTION

Since ancient times, binding agreements that recognize and manage the rights and duties between parties have been consummated. These binding agreements, in which an exchange of value is made, are now known as contracts and serve as protection for both parties. These contracts are written or spoken and require the parties to trust each other to fulfil their side of the commitment. The massively complex set of contractual agreements that are created, due to the way society is structured, lead to enforce trust by centralized organizations like banks. Making the same society, dependent upon third parties to manage and enforce those contractual agreements.

In (Nakamoto, 2009) proposed a peer-to-peer network where transactions are hashed into an ongoing chain, forming a record that cannot be changed without redoing the Proof-of-Work, naming it BC. Later on, a BC called Ethereum was specifically created and designed to support SCs, that allowed to create contracts between parties. Nowadays, SCs are enforced by the network of computers that makes up the BC, guaranteeing the adequate execution of the code that composes the SCs, based on proven cryptographic algorithms. These new technologies are undeniable faster, cheaper and more secure than many traditional systems, which helps with the handling of this massive amount of complex contractual agreements and transactions that are nowadays created. However, building systems on BC is non-trivial due to the steep learning curve of the BC technology.

Model-driven engineering tools can generate well-tested code implementing best practices and help developers manage software complexity by only focusing on building high-level models without requiring expert development knowledge (Schmidt, 2006). Thus, it would be expected that when using Model-driven engineering tools to produce BC SCs this would decrease the slope of the learning curve, making the production faster.

As explained in (Norta, 2017), referencing a crowdfunding project that was hacked as it contained security flaws, resulting in a considerable monetary loss. *"The incident shows it is not enough to merely equip the protocol layer on top of a Blockchain with a Turing-complete language such as Solidity to realize smart-contract management. Instead, we propose in this keynote paper that is crucial to address a gap for secure smart-contract management pertaining to the currently ignored application-layer development"*.

A solution to this referred problem would be to generate SCs automatically, which would add a level of security. However, as novelties, these entail ab-

Aparício, M., Guerreiro, S. and Sousa, P.
Towards an Automated DEMO Action Model Implementation using Blockchain Smart Contracts.
DOI: 10.5220/0009417907620769
In *Proceedings of the 22nd International Conference on Enterprise Information Systems (ICEIS 2020) - Volume 2*, pages 762-769
ISBN: 978-989-758-423-7

sence of formal models and systematization (de Kruijff and Weigand, 2017b). To deal with such degree of ambiguity and inconsistency Ontology seems a good fit to deal with the automation task of creating SCs (Guarino et al., 2009).

The main goals for this paper derive from the fact that there are already papers (Hornáčková et al., 2019), (de Kruijff and Weigand, 2017b) that suggest the automatic generation of SCs resorting to DEMO Ontology, but they don't prove its sufficiency or even apply them to a sizable sample of real SCs. This paper aims to infer if by using Ontology, more specifically the DEMO Ontology and the DEMO Action Model, it is possible to extract automatically the knowledge required to generate SCs on BC, allowing it to ensure the correctness of these same SCs.

The main contributions of this paper, considering the main question, are: 1) Determine whether Ontology builders are sufficient for the automatic generation of SCs; If insufficient, the missing knowledge will have to be acquired; If sufficient, with that knowledge proceed; 2) Through the knowledge acquired, proceed to formalize principles of translating DEMO Action Models into contract code; 3) Make the creation of SCs more accessible to whom may desires.

2 BACKGROUND

2.1 Blockchain

BC was originally described in 1991, by a group of researchers and was originally intended to timestamp digital documents so that it was impossible to tamper them (Haber and Stornetta, 1991). It went by mostly unused until it was adapted by (Nakamoto, 2009), to create the digital cryptocurrency Bitcoin. Nakamoto described BC as an architecture that gives participants the ability to perform electronic transactions without relying on trust. It started as the usual framework of coins made from digital signatures, which allows strong control of ownership. But still required a trusted third party to prevent the Double-Spending Problem, which was solved by a peer-to-peer network approach. This network timestamps transactions by hashing them into an ongoing chain of hash-based Proof-of-Work, forming a record that cannot be changed without redoing the Proof-of-Work.

What makes this possible it that, each block contains some data, the hash of the block and the hash of the previous block. The data that is stored inside a block depends on the type of BC, but normally stores the details of multiple transactions, each with an identification for the sender, the receiver and the asset. A block also has a hash that identifies its content and it's always unique. If something is changed inside a block, that would cause the hash to change. That's why hashes are very useful to detect changes in blocks. The hash of the previous block effectively creates a chain of blocks and it's this technique that makes a BC so secure. However, the hashing technique is not enough, with the high computational capacity that exists today, where a computer as the capacity of calculate hundreds of thousands of hashes, per second. To mitigate this problem, BC has a consensus mechanism called Proof-of-Work. This mechanism slows down the creation of new blocks since, if a block is tempered the Proof-of-Work of all the previous blocks has to be recalculated. So, the security of BC comes from its creative use of hashing and a Proof-of-Work mechanism. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll outpace attackers. Of course, its distributive nature also adds a level of security, since instead of using a central entity to manage the chain, BC uses a peer-to-peer network where anyone can join. Information is broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest Proof-of-Work chain as proof of what happened while they were gone.

2.2 Smart Contract

(Szabo, 1997) , long before Bitcoin was created, drew up the term *"Smart Contracts"* and explains them as follows: *"A smart contract is a set of promises, specified in digital form, including protocols within which the parties perform on these promises"*.

In the context of BC, in particular second-generation BC, SCs are just like contracts in the real world. The only difference is that they are completely digital, in the sense that they are both defined by software code and executed or enforced by the code itself automatically without discretion. The trust issue is also addressed, once SCs are stored on a BC, they inherit some interesting properties: immutable and distributed. Being immutable means that once a SC is created, it can never be changed again. So, it isn't possible to tamper the code of the contract. Being distributed means that the output of the contract is validated by every node on the network. So, a single node cannot force the behavior of the contract since it is dependent of the other nodes. Like all algorithms SCs may require input values, and only act if certain predefined conditions are met. When a particular value is reached the SC changes its state and executes the functions, that are programmatically predefined algorithms, automatically triggering an event on the BC.

If false data is inputed to the system, then false results will be outputed (Bahga and Madisetti, 2016).

2.3 Enterprise Ontology

In order to really cope with the current and the future challenges, of the social and economic structure, conceptual models of the enterprises are needed. These models are known as ontological models, and most comply with these five properties: *coherence*, *comprehensiveness*, *consistency*, *conciseness*, and *essence*, collectively abbreviated as C_4E. By *coherence* its mean that the distinguished aspect models constitute a logical and truly integral whole. By *comprehensive* its mean that all relevant issues are covered, that the whole is complete. By *consistent* its mean that the aspect models are free from contradictions or irregularities. By *concise* its mean that no superfluous matters are contained in it, that the whole is compact and succinct. The most important property, however, is that this conceptual model is *essential*, that it shows only the essence of the enterprise. In particular, it means that the model abstracts from all realization and implementation issues (Dietz, 2006).

A complete enterprise ontology consists of four related aspect models. The *Construction Model* it is the most concise model, and specifies the identified transaction types and the associated actor roles (with specific authority and responsibility, but not a specific person), as well as the information links. The *Process Model* contains, for every transaction type in the *Construction Model*, the specific transaction pattern of the transaction type. It also contains the causal and conditional relationships between transactions. The *Action Model* specifies the action rules that serve as guidelines for the actors in dealing with their agenda. The *Fact Model* specifies the state space of both the production world and the coordination world of the enterprise.

With Ontology it's possible to make a common interpretation of data across organizations, making it easier and cheaper to exchange transactions.

2.4 DEMO Theory

DEMO, is an enterprise modelling methodology for (re)designing and (re)engineering organizations. In DEMO, an enterprise is seen as a system of people and their relations, authority and responsibility. The usage of a strongly simplified models that focus on people forms the basis of DEMO. By using a language that is common in the enterprise, the understanding of such models are guaranteed, even though they're abstract and have a conceptual nature. DEMO describes the construction and operation of the organisation, while completely abstracting from implementation and realisation. The core concept of DEMO is a transaction and is fully based on the $\psi - theory$.

According to $\psi - theory$, in the standard pattern of a business transaction exists two actor roles, the initiator and the executor. The obtained fact when performing a business transaction, is originated by the collaboration of production and coordination acts. These acts contain three phases each one with specific steps. (1) The Order phase that contains the request (rq); promise (pm); decline (dc); quit (qt) steps. (2) The Execution phase that contains only the execution (ex) step. (3) The Result phase that contains the states (st); reject (rj); accept (ac) steps. The following four steps are present in every transaction and represent the happy flow: request, promise, state, accept. Each of these steps can be revoked (rv) anytime during the transaction. The other party can either allow (al) this, or refuse (rf) it. The $\psi - theory$ is supported by four axioms. The *Operation Axiom* abstracts from the subjects in order to focus on the different actor roles they fulfill. An actor is a subject fulfilling an actor role. Actors perform two kinds of acts: production acts and coordination acts. By performing production acts they contribute to achieving the purpose or the mission of the enterprise. By performing coordination acts they enter into and comply with mutual commitments about production acts. The *Transaction Axiom* states that production and coordination acts occur in patterns, called transactions. The *Composition Axiom* states that every transaction is either enclosed in some other transaction or it is a customer transaction or it is a self-activating transaction. At last, the *Distinction Axiom* is about the integrating role that human beings play in constituting an enterprise. Three human abilities are distinguished, called *performa*, *informa*, and *forma*.

As a methodology, DEMO has four models already mentioned: the *Construction Model*, the *Process Model*, *Action Model* and the *Fact Model*.

3 RELATED WORK

The work of (Kim and Laskowski, 2016) was considered since, it believed that ontologies could contribute to develop BC applications. In fact, argued that ontology-based BC modeling would result in a BC with enhanced interpretability, stating even that *"A modeling approach based on formal ontologies can aid in the formal specifications for automated inference and verification in the operation of a blockchain"*. This only reinforced the initial idea

of this work, where it is believed that a modeling approach based on formal ontologies can aid in the development of SCs that execute on the BC. Going further (Kim and Laskowski, 2016) even a Proof-of-Concept did, where the translation of TOVE Traceabiliy Ontology axioms into SCs that could execute a provenance trace and enforce traceability constraints on the BC.

"Towards a Blockchain Ontology" (de Kruijff and Weigand, 2017a) and *"Understanding the Blockchain Using Enterprise Ontology"* (de Kruijff and Weigand, 2017b), used Enterprise Ontology and DEMO to describe the BC Ontology from a Datalogical, Infological and Essential (Business) perspective. This papers suggests that by specifying the BC application on the business level first, it will be possible to generate the BC implementation automatically, with some design parameters to be set.

In particular, (de Kruijff and Weigand, 2017b) highlights the distinction axiom as highly relevant for the BC. Following the three abilities already mentioned in section 2 three ontological layers are distinguished. Starting from the Datalogical layer that describes BC transactions at the technical level in terms of blocks and code. From there, an Infological abstraction is done, in order to describe the BC transactions as effectuating an (immutable) open ledger system. This layer aims to abstract from the various implementations that exist today or will be developed in the future. To describe the economic meaning of the Infological transactions the Essential layer is used. This last, is the preferred level of specification for a BC application as it abstracts from the implementation choices. An overview is provided for the UML classes that are used to construct the domain Ontology for BC at the Datalogical abstraction layer. The subject matter in this paper are SCs, so it should be noted that these are seen as a type of transaction, at this level of abstraction. In regards to the Infological level, the SCs are enforced by rules of engagement that are implemented as BC code. At last, for the Essential level, a contract is an agreement between agents consisting of mutual commitments. A SC is a contract in which the commitment fulfillment is completely or partially performed automatically. In fact, Kruijff et al. didn't validate their Ontology and the focal point was the BC as a whole.

There have been attempts at raising the level of abstraction from code-centric to model-centric SCs development. The different approaches tried so far, can be divided into three: the Agent-Based Approach, the State Machine Approach and the Process-Based Approach. The Agent-Based Approach described by (Frantz and Nowostawski, 2016) proposes a modeling approach that supports the semi-automated translation of human-readable contract representations into computational equivalents in order to enable the codification of laws into verifiable and enforceable computational structures that reside within a public BC. They identify SC components that correspond to real world institutions, and propose a mapping using a domain-specific language in order to support the contract modeling process. The concept Grammar of Institutions (Crawford and Ostrom, 1995) is used to decompose institutions into rule-based statements. These statements are then compiled in a structured formalization. In this case, the statements are constructed from five components, abbreviated to ADICO. The **A**ttributes describing an actor's characteristics or attributes. The **D**eontic describing the nature of the statement as an obligation, permission or prohibition. The A**I**m describing the action or outcome that this statement regulates. The **C**onditions describing the contextual conditions under which this statement holds. And the **O**r else describing consequences associated with non-conformance. Using these components, statements on the execution of the smart contract are made. The statements are then linked by the structure of Nested ADICO (Boella et al., 2013), a variant of ADICO in which the institutional functions are linked by the operators AND, OR, and XOR to create a simple set of prescriptions. The set of prescriptions is then transformed into a contract skeleton which has to be finished manually. Furthermore, it is argued that the Grammar of Institutions invites non-technical people to the SC development process.

The State Machine Approach is based on the observation that SCs act as state machines. A SC is in an initial state and a transaction transitions the contract from one state to the next. The possibility of SCs as state machines is also described in the Solidity specification. (Mavridou and Laszka, 2018) show that the transformation of the Finit State Machine to Solidity is partly automated, since to ensure Solidity code quality, some manual coding might be necessary or added though plugins.

For Process-Based Approaches, both DEMO and BPMN are well established for modelling business processes. (Weber et al., 2016), describes a proposal to support inter-organizational processes through BC technology. Weber et al. developed a technique to integrate BC into the choreography of processes in order to maintain trust. The BC enabled to store the status of process execution across all involved participants, as well as to coordinate the collaborative business process execution. Validation was made against the ability to distinguish between conforming and non-conforming traces. (García-Bañuelos

et al., 2017) presents an optimization for (Weber et al., 2016). In this work, to compile BPMN models into a SC in Solidity Language, the BPMN model is first translated into a reduced Petri Net. Only after this first step, the reduced Petri is compiled into a Solidity SC. Compared to (Weber et al., 2016) this work (García-Bañuelos et al., 2017) managed to decrease the amount paid of resources and achieve a higher throughput. Caterpillar, first presented in (Pintado, 2017) and further discussed in (Pintado et al., 2018), is an open-source Business Process Management System - BPMS - that runs on top of the Ethereum BC. Like any BPMS, Caterpillar supports the creation of instances of a process model (captured in BPMN) and allows users to track the state of process instances and to execute tasks thereof. The specificity of Caterpillar is that the state of each process instance is maintained on the Ethereum BC, and the workflow routing is performed by SCs generated by a BPMN-to-Solidity compiler. Given a BPMN model (in standard XML format), it generates a SC (in Solidity), which encapsulates the workflow routing logic of the process model. Specifically, the SC contains variables to encode the state of a process instance, and scripts to update this state whenever a task completes or an event occurs. Caterpillar supports not only basic BPMN control flow elements (i.e. tasks and gateways), but also includes advanced ones, such as subprocesses, multi-instances and event handling. (Tran et al., 2018), on other hand can automatically create well-tested SC code from specifications that are encoded in the business process and data registry models based on the implemented model transformations. The BPMN translator can automatically generate SCs in Solidity from BPMN models while the registry generator creates Solidity SC based on the registry models. The BPMN translator takes an existing BPMN business process model as input and outputs a SC. This output includes the information to call registry functions and to instantiate and execute the process model. The registry generator takes data structure information and registry type as fields, and basic and advanced operations as methods, from which it generates the registry SC. Users can then deploy the SCs on BC. This work builds up on already seen works, such as (García-Bañuelos et al., 2017) (Weber et al., 2016), for the BPMN translation algorithms. However, BPMN doesn't adequately support the articulation of business rules, that would be essential for the automatic generation of SCs. Process modellers normally, have to use workarounds, as additional tools, to include business rules in their processes.

Lastly, (Hornáčková et al., 2019), proved to be a really important related work, due to its complete alignment with the purpose of this work. Actually its *Further Research* section, makes reference to one of the things this work is trying to reach: formalization of principles of translating DEMO models into contract code. In (Hornáčková et al., 2019) work, an IT system based on Enterprise Engineering integrating SCs is proposed. Also, a Proof-of-Concept implementation of a SC of a mortgage process using a DEMO methodology was developed, to demonstrate the feasibility of the proposed concepts. Proving this way that BC SCs can be used in the implementation of an enterprise information system (EIS) based on DEMO methodology. Further, shares the same believe that by applying the DEMO methodology a reduction of unwanted states, the prevention of errors and a improvement of security, which is crucial for SCs, since they are representing valuable assets, can be reached. However, never formalizes the principles or even applies this beliefs to a sizable sample of use cases.

4 PROPOSED SOLUTION

4.1 Ontology for Smart Contracts

The concept of data independence designates the techniques that allow data to be changed without affecting the applications that process it (Markov, 2008). It is the ability to modify a scheme definition in one level without affecting a scheme definition in a higher level. It is believe that a similar separation is highly needed for SCs, in order to achieve the goals set in this work. DEMO is based on explicit specified axioms characterized by a rigid modeling methodology, and is focused on the construction and operation of a system rather than the functional behavior. It emphasizes the importance of choosing the most effective level of abstraction during information system development, in order to establish a clear separation of concerns. The adoption of the *Distinction Axiom* of Enterprise Ontology, presented in section 2, is proposed as an ontological basis for this separation.

In a first approach to the problem, it is believed that, for the Datalogical Layer a SC can be defined as a piece of code contained on a node of the BC. For the Infological layer a SC is enforced by a set of rules implemented on the BC through code. And lastly, for the Essential layer, the SC is a contract in which the commitment fulfillment is completely or partially performed automatically in BC.

This step allows to defend Ontology, and in particular DEMO, as a good way towards a model-driven

approach in regards to the automatic generation of software artifacts. However doesn't make it achievable.

4.2 Automatic Extraction Process

The work that is here trying to be achieved is very much aligned with the one Hornackova, Skotnica and Pergl did in *"Exploring a Role of Blockchain Smart Contracts in Enterprise Engineering"* (Hornáčková et al., 2019), to some extent. Their workwas applied to a financial transaction and so as our will be for a more trivial parallelism. The chosen financial transaction is a mortgage process. With this use case the advantages and compatibilities between DEMO and SCs are going to be presented. It is composed of 14 action rules all with a similar structure to the struture presented on figure 1.

WHEN	mortgage completion for new Mortgage is requested (rq)	
	with	the receiver of Mortgage is Client the property of Mortgage is Property
ASSESS	**justice:**	the performer of the request is the receiver of Mortgage
	sincerity:	<no specific conditions>
	truth:	Mortgage for property for receiver is approved
IF	complying with request is considered justifiable	
THEN	promise mortgage completion of Mortgage (pm)	
ELSE	decline mortgage completion for Mortgage (dc)	

Figure 1: The Action Rule 1, for the Mortgage Use Case.

Both works start with a believe that a DEMO transaction is represented as a contract in a BC. The contract has its own address, internal storage, attributes, methods and it is callable by either an external actor or another contract, as mentioned in section 2. This is the functionality needed to represent a DEMO transaction. However, the divergence between the work made in (Hornáčková et al., 2019) and this is in how the automatic extraction process is done.

Now, this present work argues that SCs automated generation could be done directly form the Action Model with no need for the generation of the remaining DEMO models. It believes that the structure and content of a SC can be directly mapped to each action rule. Since that by creating the action rules it is also being created the logic on which the SCs operate. This change presents a clear improvement over the process presented by (Hornáčková et al., 2019) considering that with this new process the SC creator does not have to have any knowledge of the DEMO methodology.

In fact, the action rules contain all the decomposed detail of the above models, in fact the basis of the DEMO methodology is exactly the *Action Model*, as can be seen in figure 2. The *Construction Model* specifies the construction of the organization, specifies the identified transaction types and the associated actor roles, as well as the information links between the actor roles and the information banks. By occupying the top of the triangle it is suggested that is the most concise model. The *Process Model* contains, for every transaction type in the *Construction Model*, the specific transaction pattern of the transaction type. And, also contains the causal and conditional relationships between transactions. The *Process Model* is put just below the *Construction Model* in the triangle because it is the first level of detailing of the *Construction Model*, namely, the detailing of the identified transaction types.

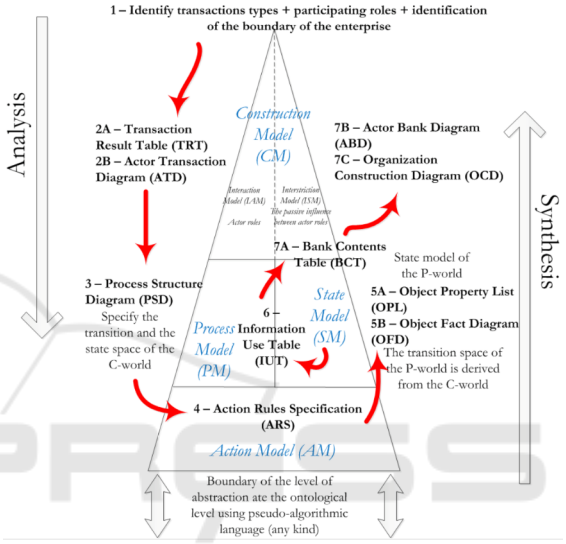

Figure 2: DEMO Methodology, from (Guerreiro, 2012).

The *Action Model* specifies the action rules that serve as guidelines for the actors in dealing with their agenda. The *Action Model* is put just below the *Process Model* in the triangle because it is the second level of detailing of the *Construction Model*, namely, the detailing of the identified steps in the *Process Model* of the transaction types in the *Construction Model*. At the ontological level of abstraction there is nothing below the *Action Model*. The *Fact Model* is put on top of the *Action Model* in figure 2 because it is directly based on the *Action Model*; it specifies all object classes, fact types, and ontological coexistence rules that are contained in the *Action Model*. The *Action Model* is in a very literal sense the basis of the other aspect models since it contains all information that is (also) contained in the *Construction Model*, *Process Model*, and *Fact Model*; but in a different way. The *Action Model* is the most detailed and comprehensive aspect model. It's like the other three aspect models are derived from the *Action Model*. The language and processes from which an *Action Model* is created, are simple. Normally, an *Action Model* is represented in Action Rule Specifications and Work

When	<insert Attribute>
is	<insert Deontic>
in case	<insert Conditions>
than it is	<insert Aim>
otherwise	<insert Or else>

Figure 3: The Proposed Template for Specifying Action Rules.

Instruction Specifications. This two specifications are human-readable behaviour specifications, with a standard structure.

The proposed task would be to translate formalizations from a human-readable behaviour specification, in this case action rules, to a contractual structure in the form of a SC. To do so, it will be used a *Grammar of Institutions* (Crawford and Ostrom, 1995), that captures essential institutional characteristics, as DEMO does. This idea comes from (Frantz and Nowostawski, 2016) work, already mention. With this representation it is possible to decompose specifications into simple rule-based statements. These statements are constructed with the five components (ADICO): **A**ttributes, **D**eontic, A**I**m, **C**onditions and **O**r else, described more in detail in section 3. Taking as an example the first action rule defined for the mortgage process, in figure 1. The Blue color represents the **A**ttribute component. The color Green represents the **D**eontic. The color Magenta represents the **C**onditions. The color Orange represents the A**I**m component. And at last, the color Pink represents the **O**r else component. To make the detection of each element simpler a template is proposed in figure 3. Where the **D**eontic, A**I**m and **O**r else components can have one of the following values: request (rq), promise (pm), decline (dc), quit (qt), state (st), reject (rj) and accept (ac). Only the A**I**m and **O**r else conditions can have the execute (ex) value. To decompose complex prescription into simpler ones the Nested ADICO (Boella et al., 2013) may be used.

Mapping these components into structures supported by the Solidity language can be done. The **A**ttributes components are structures; the **D**eontic components are equivalent to function modifiers; the A**I**m components are mapped though functions and events; the **C**onditions though function modifiers; and at last the **O**r else components though throw statement or alternative control flow.

5 WORK EVALUATION METHODOLOGY

Further, validation is to be done with applications as well as by establishing mappings to existing implementations of BC. This work has two main components. A more theoretical one, where is necessary to prove the sufficiency of ontological concepts to represent a SC. And the other component where the automatic generation of a SC from a DEMO Action Model will be attempted. For the theoretical component an ontological validation will be performed. Both *"Validating domain ontologies: A methodology exemplified for concept maps"* (Steiner and Albert, 2017) and *"Ontological Evaluation and Validation"* (Tartir et al., 2010) will be used. However, it is important to ensure that this possible validity applies to any and all SC. For this will be used a database containing SCs from real contexts.

As with any programming task, the human-readable contract and associated obligations need to be codified, and then subsequently verified, to ensure that the machine-readable representation, in this case Solidity, conforms to specified behaviour. For the more practical component, where the automatic generation of a SCs from a DEMO Action Model will be attempted, validation will be based on a tool (http://smartcontracts.azurewebsites.net/) presented on *"Exploring a Role of Blockchain Smart Contracts in Enterprise Engineering"* (Hornáčková et al., 2019). This tool already generates SC from DEMO models, as detailed in section 3.To ensure that this mapping between DEMO Action Models and SCs is possible for any and all SC the already mentioned database will be used.

6 CONCLUSIONS

Non-technical people, that do not comprehend software code are once again dependent on a third party to write them contracts. The low level of semantics of software code makes it challenging to have a high level of comprehension and reasoning, which makes it more prone to errors. There are already some research towards raising the level of abstraction from code-centric to model-centric SCs development. However, either unintuitive processes are created that do not meet the goal of facilitating development, or use models that are not very commendable because of their nature, such as BPMN.

To address this problem we propose a template with a simple structure, and where some elements are strict to predefined values, that allows to model and develop a SC in Solidity. With this template it will be more intuitive for non-technical people to develop their own SC without needing third-parties. To date, an Ontology for SC is proposed for all layers: Datalogical, Infological and Essential, although the latter is the most recommended to achieve the intended

purpose. With the validation of the Ontology it will be possible to verify other works similar to this, that already use Ontology but don't prove it's sufficiency and support the idea of extracting SCs from Ontological models such as the Action Model. Furthermore, a template, based on action rules structure, is proposed and a mapping between the templates elements and Solidity language elements is proposed.

For future work, the Ontologial validation will be attempt, if Ontology proves to be sufficient an software artefact will be implemented. This prototype will allow from a DEMO Action Model extract a SC written in Solidity.

ACKNOWLEDGEMENTS

This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020 and by the European Commission program H2020 under the grant agreement 822404 (project QualiChain).

REFERENCES

Bahga, A. and Madisetti, V. (2016). Blockchain platform for industrial internet of things. *Journal of Software Engineering and Applications*, 09:533–546.

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

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

de Kruijff, J. and Weigand, H. (2017a). Towards a blockchain ontology.

de Kruijff, J. and Weigand, H. (2017b). Understanding the blockchain using enterprise ontology.

Dietz, J. L. G. (2006). *Enterprise Ontology: Theory and Methodology*. Springer-Verlag, Berlin, Heidelberg.

Frantz, C. K. and Nowostawski, M. (2016). 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.

García-Bañuelos, L., Ponomarev, A., Dumas, M., and Weber, I. (2017). Optimized execution of business processes on blockchain.

Guarino, N., Oberle, D., and Staab, S. (2009). *What Is an Ontology?*, pages 1–17.

Guerreiro, S. (2012). Enterprise dynamic systems control enforcement of run-time business transactions using demo: principles of design and implementation. Instituto Superior Técnico-Universidade Técnica de Lisboa, Lisboa.

Haber, S. and Stornetta, W. S. (1991). How to time-stamp a digital document. *J. Cryptol.*, 3(2):99–111.

Hornáčková, B., Skotnica, M., and Pergl, R. (2019). Exploring a role of blockchain smart contracts in enterprise engineering. In Aveiro, D., Guizzardi, G., Guerreiro, S., and Guédria, W., editors, *Advances in Enterprise Engineering XII*, pages 113–127, Cham. Springer International Publishing.

Kim, H. and Laskowski, M. (2016). Towards an ontology-driven blockchain design for supply chain provenance.

Markov, K. (2008). Data independence in the multi-dimensional numbered information spaces.

Mavridou, A. and Laszka, A. (2018). *Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach*, pages 523–540.

Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*.

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

Pintado, O. (2017). Caterpillar: A blockchain-based business process management system.

Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., and Ponomarev, A. (2018). Caterpillar: A business process execution engine on the ethereum blockchain.

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31.

Steiner, C. M. and Albert, D. (2017). Validating domain ontologies: A methodology exemplified for concept maps. *Cogent Education*, 4(1):73–82.

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

Tartir, S., Arpinar, I., and Sheth, A. (2010). *Ontological Evaluation and Validation*, pages 115–130.

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

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