Blockchain Smart Contract Meta-modeling

N. Sánchez-Gómez[1,*], J. Torres-Valderrama[1],
Manuel Mejías Risoto[1] and Alejandra Garrido[2]

[1]*Web Engineering and Early Testing Research Group, ETSII, University of Seville, Spain*
[2]*LIFIA, Fac. de Inform., Univ. Nac. de La Plata & CONICET, Argentina*
E-mail: nicolas.sanchez@iwt2.org; jtorres@us.es; risoto@us.es; garrido@lifia.info.unlp.edu.ar
[*]*Corresponding Author*

Received 31 March 2021; Accepted 22 July 2021;
Publication 22 October 2021

Abstract

One of the key benefits of blockchain technology is its ability to keep a permanent, unalterable record of transactions. In business environments, where companies interact with each other without a centralized authority to ensure trust between them, this has led to blockchain platforms and smart contracts being proposed as a means of implementing trustworthy collaborative processes. Software engineers must deal with them to ensure the quality of smart contracts in all phases of the smart contract lifecycle, from requirements specifications to design and deployment. This broad scope and criticality of smart contracts in business environments means that they have to be expressed in a language that is intuitive, easy-to-use, independent of the blockchain platform employed, and oriented towards software quality assurance. In this paper we present a key component: a first outline of a UML-based smart contract meta-model that would allow us to achieve these objectives. This meta-model will be enriched in future work to represent blockchain environments and automated testing.

Keywords: Smart contract, model-based, meta-model, UML.

Journal of Web Engineering, Vol. 20_7, 2059–2080.
doi: 10.13052/jwe1540-9589.2073
© 2021 River Publishers

1 Introduction

Blockchain is now one of the most transformative technological developments of our time. Basically, it is a form of distributed ledger: i.e., a transactions log that ensures immutability and verifiability [1].

A blockchain transaction typically contains a predefined set of metadata and an optional payload grouped into chronologically concatenated ”blocks” which are linked together securely and immutably using cryptographic techniques. Thanks to blockchain’s intrinsic mechanisms for facilitating both external and internal audits [2], traceability is therefore one of this technology’s biggest advantages over other solutions.

Smart contracts are programs deployed and run on blockchain networks. These programs extend the functionality of a blockchain and allow untrusted parties to establish trust in the truthful execution of an agreement [3].

Smart contracts for industries, public sector, financial institutions, etc. require external off-chain data such as IoT (Internet of Things) data, citizenship data or stock values to trigger execution. In the sectors mentioned, it is also necessary to integrate business processes with blockchain networks [4]. Thus, the use of smart contracts is essential to his integration, since, from an external viewpoint, the public function of smart contracts constitutes the blockchain integration points.

There are currently numerous blockchain platforms: Ethereum, focuses on the capability of automatic digital asset management, supports smart contracts and also adapts the PoW consensus protocol; Hyperledger, an open source collaborative effort created to advance permissioned or private, cross-industry blockchain technologies; MultiChain, a platform to create and deploy permissioned or private blockchain networks; and an extensive list of platforms [5, 6].

In this blockchain platforms, smart contracts are invoked using different protocols, techniques, and data formats, although their implementation is like that of a Class in any object-oriented programming language (OOPL). These peculiarities, together with typical human error, make the development and integration process error-prone. Smart contracts therefore need to be designed using best practices that reduce the number of coding and operational errors [7].

For some years now, model-driven engineering and modelling tools have helped to document the functionality of business processes and automatically generate software source code through model transformations. Unified Modelling Language (UML) and other modelling standards are used for

this purpose. Models are typically platform-independent [8] and easier to understand than software source code [9].

Thus, the use of models improves development productivity and quality. Moreover, modelling tools can ensure that the deployed code has not been modified after its generation from the model [11]. But to achieve this objective, a good way is to realize a software development approach based on the concepts of metamodel and model and model-to-model transformations. With this approach, source models are transformed into target models to generate, automatically or semi-automatically, the final executable source code during the software development process.

With this in mind, and given the immutability of blockchain technology, it is essential that software be fully evaluated and validated before a smart contract code is deployed in a blockchain network. A defect in the code could have an irreparable effect on the blockchain network, and this paper proposes a way to minimize this impact through the use of Blockchain smart contract meta-modeling.

The rest of this paper is organized as follows, Section 2 describes some relevant background. In Section 3, we then analyze the structure of smart contracts in detail. Section 4 includes research articles that propose, apriori, the use of smart contract metamodels and their contributions. Section 5 presents our definition of an initial approach to a meta-model capable of describing smart contracts independently of the blockchain platform. The paper ends in Section 6, with a discussion of our proposal and ideas for future work.

2 Background: Blockchain Smart Contract, Its Anatomy, and Other Related Topics

This section introduces the most important concepts used throughout this paper. First, the basic characteristics of a blockchain smart contract are outlined. It then reviews the general characteristics and anatomy of smart contracts, before going on to conceptualize model-based software engineering and the need for quality assurance in smart contract software, together with the benefits this could bring.

2.1 Blockchain Smart Contract

A blockchain is a distributed peer-to-peer system of ledgers. It uses a software unit comprising an algorithm, which negotiates the informational content of

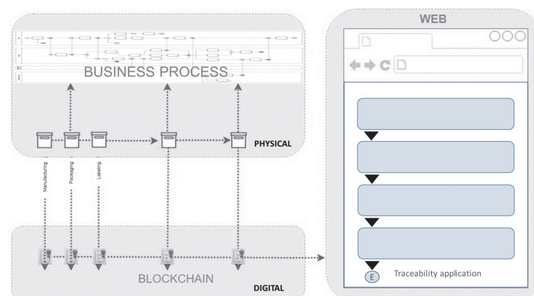

Figure 1 Supply chain traceability system for blockchain technology [19].

ordered connected blocks of data, also employing cryptographic and security technologies to achieve and maintain its integrity [10].

A smart contract is a computer program which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a contract or an agreement [13].

Smart contracts are supported on many blockchain platforms (e.g., Ethereum [14], Hyperledger [15], etc.), but may be very limited on others (e.g., Bitcoin [16]). They extend the functionality of a blockchain and allow untrusted parties to establish trust in the truthful execution of the agreement [3]. In short, smart contracts are programs deployed and run on blockchain networks, and are capable of executing triggers, rules, and business logic to enable transactions [17].

Smart contracts must be deployed and instantiated on a blockchain network, and this process creates an instance of the contract and initializes its state. After this initialization, the computer program becomes accessible to possible clients who can invoke the contract via its external interface, by submitting suitable transactions that carry the invocation in their body. Invocations may come from other smart contracts inside the same blockchain or from the outside. The exact way in which smart contracts are invoked is, again, dependent on the blockchain platform used.

To integrate business processes with blockchain networks [4], it is essential to use smart contracts since, from an external viewpoint, the public functions of smart contracts are the access points where blockchains to be used by other systems: i.e., they constitute the blockchain integration points.

To illustrate this concept, Figure 1 shows a supply chain traceability system for blockchain technology. Supply chain is the connection of all business processes involved in the commercialization, generation, and distribution of goods, from raw materials to finished products and end consumers [18].

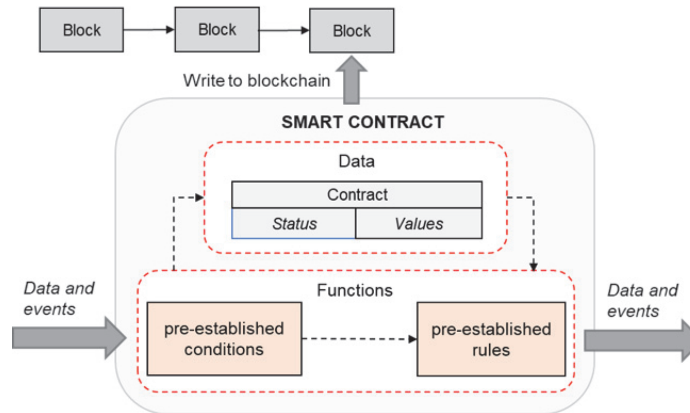

Figure 2 Smart contract anatomy.

Given the large number of blockchains, each one invoking a smart contract using different protocols, techniques, and data formats, significantly raising raises the integration barrier for systems wishing to use them. Software engineers must be aware of these variations which make the integration process prone to error.

2.2 Smart Contract Anatomy

Smart contract is a computer program which, once deployed, is run at a given address on a blockchain network. Smart contracts are made up of **Data** and **Functions**, as shown in Figure 2, that can be executed upon receipt of a transaction [20]. During execution smart contracts can also send messages to other contracts. These messages comprise the sender's address, the recipient's address, the transfer value, and a data field containing the input data for the recipient contract. There is a difference between message and transaction: a transaction is produced by an external owned account (EOA) or a simple account, while a message is produced by a smart contract.

There follows an overview of what a smart contract is (see Figure 2):

Data (Contract Status / Contract Values) Any data in the contract must be assigned to a location, either *memory* or *storage* (modifying storage in a smart contract is costly, so it is necessary to analyse in detail where the data is to be housed):

Storage Persistent data is called storage and is represented by state variables. These values are permanently stored on the blockchain.

Memory Values that are stored only during the execution of a contract function are called memory variables. Since these are not stored permanently on the blockchain network, they are much cheaper to use.

In addition to these variables, there also exist some special global variables that are primarily used to provide information about the blockchain or the current transaction.

Functions (pre-established conditions and rules) Functions basically obtain or establish information in response to incoming transactions.

There are two basic types of functions: (i) internal, which are the functions that can access state variables, and (ii) external, which are part of the smart contract interface, meaning that they can be called from other smart contracts and through transactions.

Functions can also be: (i) public functions, which can be called internally from within the contract or externally through messages, and (ii) private functions, which are only visible for the smart contract they are defined in and not in derived contracts. Both functions and state variables can be made public or private. Finally, there is a specific function, called constructor functions, which is executed only once, when the contract is first deployed. Like constructors in class-based programming languages, this functions typically initialise state variables to their specified values.

2.3 Model-Driven Software Engineering for Blockchain Smart Contracts

Model-Driven Software Engineering (MDSE) is a paradigm that uses models to address the complexity of software development at different levels of abstraction [21].

Over the past few years, modelling tools have helped to document the functionality of business processes and, through model transformations, to partially automate the generation of software source code.

With regard to blockchain smart contracts, MDSE is of particular importance for the following reasons [9, 11]: It can facilitate communication; Models are easier to understand than code, and is easier to check the correctness of a model; MDSE tools can implement best practices, generate well-tested software code, and can produce artifacts for multiple blockchain platforms, etc.

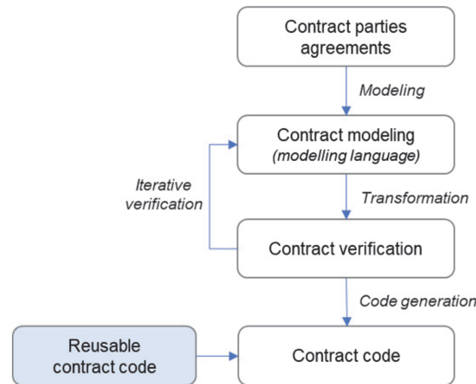

Figure 3 Model-based smart contract engineering.

2.4 Smart Contract Quality Assurance

A software defect is an error or bug in a computer program that causes it to produce an incorrect or unexpected result, or to behave in unintended and/or undesirable way. Smart contract defects are related not only to security issues, but also to design flaws which might slow down development and/or increase the risk of future bugs or errors.

Contract parties agree, sign, and fulfil contracts in accordance with the contract code. Verification and validation of this contract code is therefore essential for proper execution. It is critical to ensure that smart contracts are error-free and well-designed, before deploying them on the blockchain network. Each smart contract must go through an exhaustive quality assurance process, because any bug in the smart contract code may have an irreparable effect. If any error is found during the execution of the smart contract, a new contract must be created, since the blockchain-based contract code cannot be modified (and its data is immutable).

Model-based smart contract engineering (see Figure 3) aims to standardize the smart contract generation process and produce quality contract code. It offers the following benefits [22]: it covers all steps of contract development, early analysis and verification during system design, it eliminates repetitive low-level development work, it allows old models to be modified to obtain a new well-designed contract, etc.

In summary, the smart contract verification and validation process carried out in model-based smart contract engineering encompasses modeling, model transformation, model verification and, above all, automatic test cases and

automatic code generation, thereby facilitating the detection of potential errors as early as possible.

3 Analysis of smart contracts structure

For the implementation of smart contracts, blockchain platforms support different programming languages. Popular programming languages for blockchain development found in existing literature include Java, Python, C#, C++, Ruby, Solidity, etc. This last language, Solidity, is a new programming language designed specifically for writing Ethereum-based smart contracts.

One common misconception is that smart contracts built for Ethereum must necessarily be written in Solidity. This is not true [20]. One of the beauties of the Ethereum network and community is that you can participate using almost any programming language. Ethereum does have developer-friendly languages for writing smart contracts (Solidity and Vyper), but Ethereum and its community also embrace open source, and community projects - client implementations, APIs, development frameworks, testing tools - can be found in a wide variety of languages.

Smart contract has several components, key elements being the program code and the data storage [20]. The program code, in general, is an object that has an identity, state variables and behaviours (executable functions, events and modifiers). After reviewing different existing smart contract models (e.g., Ethereum [14], Hyperledger [15], etc.), the most important features (shown schematically in Figure 4) can be summarized as follows:

Name In addition to a unique name, the program code has a specific address corresponding to the deployment location of the contract (a smart contract is a program that runs at an address).

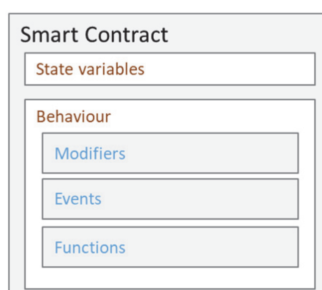

Figure 4 Smart contract basic structure [20].

State variable These are variables with global scope, meaning that they are visible (and therefore accessible) throughout the program. All these variables, return information to the contract regarding the "current" state of the blockchain. The state variables are persistent across multiple invocations. A smart contract is immutable, and its state cannot be changed after initialization (this state can be modified during the life of the contract).

Behaviour under conditions or rules:

Events and Modifiers These occur when a contract triggers an action or state change after being invoked. Normally, events have a name and parameters that represent their payload. Some blockchain platforms generate system events, others support developer defined custom events. Depending on the blockchain network, single or multiple events may be launched.

Functions These are the operations that a smart contract can perform and thus constitute its behaviour. They can usually be private or public in scope. When a function is executed, the state variables in the smart contract change depending on the logic implemented in the function. Functions all have a name, several input parameters and, optionally, output parameters. Some blockchain platforms allow direct invocation of functions using their name, while others force the use of a single dispatcher function to forward input values to target functions. Constructors are optional functions used to initialize the state variables of a contract. If no constructor is defined, a default constructor appears in the contract.

As can be deduced, then, a smart contract is like a Class in any OOPL. Figure 5 illustrates this structure with a simple example of an Ethereum smart

```
1  pragma solidity ^0.4.25;
2  contract Sample{                                   Name
3
4      //State variables
5      address private _admin;
6      uint private _state;
7
8      //Modifier
9      modifier onlyAdmin(){
10         require(msg.sender == _admin, "You are not admin");
11         _;
12     }
13
14     //Event
15     event SetState(uint value);
16
17     //Constructor
18     constructor() public{
19         _admin = msg.sender;
20     }
21
22     //Function
23     function setState(uint value) public onlyAdmin{
24         _state = value;
25         emit SetState(value);
26     }
27
28     function getValue() public view returns (uint){
29         return _state;
30     }
31
32 }
```

Figure 5 Simple Ethereum contract example.

contract. The Ethereum smart contract is divided into the following main parts:

State variable This is the backbone of the smart contract. It records the contract information. State variables are stored permanently and can be modified by functions. However, these modifications will also be included in the transaction, the update coming into effect after the transaction has been confirmed by the blockchain network.

Modifier This is used to coat the function. It is a very important part of the smart contract, because it is included in the function declaration to provide additional functions, such as checking, cleaning, etc

Event This acts like a log, recording the occurrence of an event in the blockchain network

Constructors These are used to deploy and initialize smart contracts, allowing data to be passed in and written to the state variable. Unlike Java, Solidity can only specify one constructor

Functions Functions are used to read and write state variables. Modification of the state will be included in the transaction and will take effect after being confirmed by the blockchain network. Once it takes effect, the update will be permanently saved in the blockchain ledger. In the example contract shown, there is a function with a view modifier, indicating that the function does not modify any state variables (if it tries to modify the state variable in the view function or access the state variable in the pure function, the compiler will report an error).

4 Related Works

After a detailed literature search, we analysed different articles related to the definition of smart contract meta-models, describing the approach and scope of these works, as well as the differences with the one presented in this paper. In particular, we performed selective searches with the term "smart contract" and "model", "metamodel", "model-based", "model-driven", or "MDE", and we outline the findings below.

Hu et al. [22] propose the theoretical concept of smart contract engineering (SCE) to facilitate the generation of legal smart contracts, which is the combination of software engineering, formal methods, and computational

law. The approach followed would not, in our opinion, allow the automatic generation of test cases, as it remains at a very theoretical level.

Ladleif et al. [23] also aim to pave the way for a model-based approach in the development of legal smart contracts. These authors, on the one hand, focus on reducing the potential errors and improve efficiency during the contract development process, and, on the other hand, they combine insights from literature in law and legal informatics with capabilities of existing modelling approaches and come up with a unifying model that encapsulates the essential components of legal smart contracts. This theoretical unifying model could be used as a reference for language designers aiming for a holistic representation of legal smart contracts in a model-based architecture, but it would focus only on the whole person and the whole problem as a way of finding more healthy and sustainable solutions to legal problems. From our point of view, its approach also does not allow the automatic generation of test cases, since it is oriented to the legal representation of smart contracts.

Lu et al. [24] uses an MDE approach with the idea of implementing a smart contract generation tool called Lorikeet to evaluate smart contracts in terms of feasibility, functional correctness, and cost-effectiveness. In particular, this paper focuses on a metamodel for the smart contract interface with input/output parameters and contract connection invocation parameters but leaves out the metamodel of the smart contract behavior. In our opinion, the approach followed would not allow the automatic generation of test cases, because it focuses only on the smart contract interface.

Vandenbogaerde et al. [25] present a solidity smart contract "meta-model". From our point of view, what is defined in this paper is closer to a high-level conceptual model that refers to elements contained in Ethereum smart contracts such as functions, events, etc. and possible data structures, without considering other possible platforms.

Butijn et al. [26] present an interesting meta-model of Smart contract driven business transactions. A business process consists of a collection of tasks that are performed by business partners to achieve the shared business objectives of the stakeholders. Business Process Model and Notation (BPMN) is a standard for business process modeling that provides a graphical notation for specifying business processes in a Business Process Diagram (BPD), based on a flow-charting technique very similar to activity diagrams from UML. From our point of view, the incorporation of blockchain smart contracts into business transactions can be specified in BPMN, with no need to metamodel these business rules and functions.

Skotnica et al. [27] propose a model-driven approach to create blockchain smart contracts based on a visual domain-specific language. The design of an XML-Based language class diagram is presented, and a code generation process into a blockchain smart contract is described. The approach proposed by the authors is demonstrated in a proof-of-concept model of a decentralized mortgage process in which the contract is designed, generated, and simulated in a blockchain environment, without going into smart contract meta-modeling.

All this research work is very interesting, but the approach followed by them would not allow the automatic generation of test cases. In other cases, these works are difficult to apply to industry, as they remain at a theoretical level [22, 23, 25] or contemplate a high level of detail of data sources, when these may not be available [23, 27].

5 A Conceptual Approach to Smart Contract Meta-model

5.1 Smart Contract Meta-model

As indicated above, meta-model is a model of a model, and meta-modeling is the process of generating such models. Thus, meta-modeling is the analysis, construction and development of the rules, constraints, models, and theories that are applicable and useful when modeling a predefined class.

In this section, we present an approach to a smart contract meta-model design based on the concepts introduced above. The definition of this meta-model will help us to generalize the definition of a smart contract and to specifically define each concept involved, together with any relevant relationships and constraints.

Figure 6 presents our meta-model. This meta-model is defined using a MOF (Meta-Object Facility) proposal and represented using a UML class diagram. Following the guidelines of UML, which differentiates between class types in its meta-models, green is used to refer to a behavioral class, while red refers to a structural (concept) class. In addition, the relationships between class, attribute and method have not been indicated since they are specific to UML and can be obtained from the standard.

The smart contract meta-model has three important meta-classes. The main one is smart contract. This meta-class represents the smart contract concept and is identified by two attributes, the ID and the name. The ID is an internal code that uniquely identifies the smart contract, and the name is a brief description of the contract itself. If we compare this idea with Figure 5, it matched the first part of the smart contract definition (name).

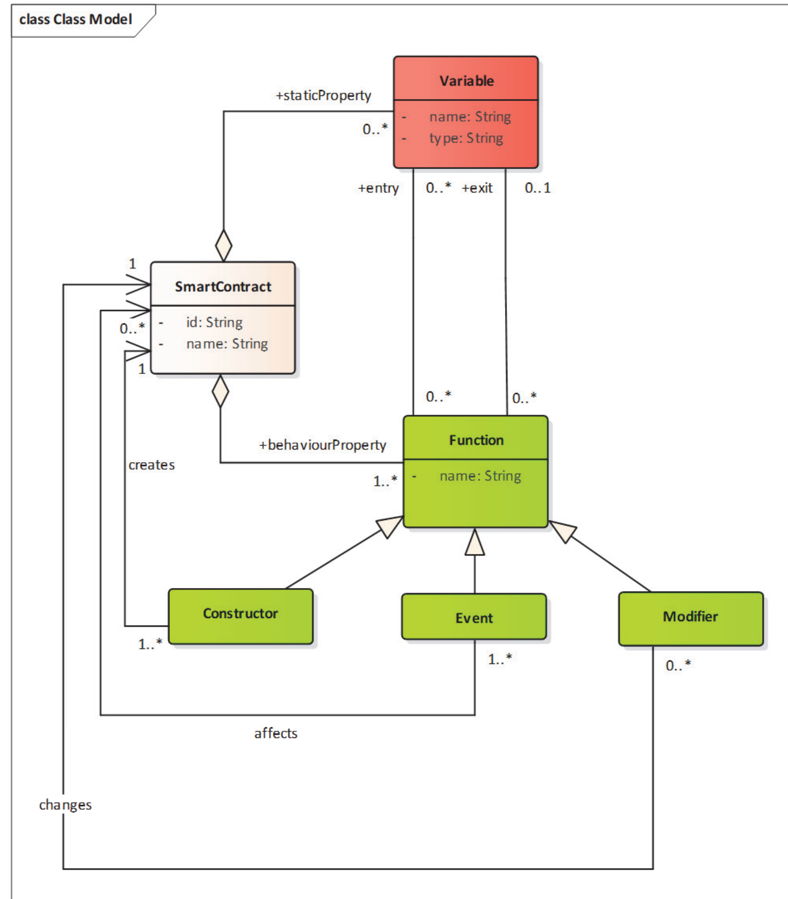

Figure 6 Smart Contract meta-model.

The other two meta-classes are Variable and Function. They were represented in the meta-model with different colours using the decorative options of UML. The Variable meta-class, coloured red (the box on top), represents the static aspect of the smart contract. The Function meta-class, coloured green (the four boxes at the bottom), represents the behavioural aspect of the smart contract. With this structure, we wanted to adhere to the same scheme that UML uses to define its concepts. The variable meta-class represents the smart contract's set of variables. It includes two attributes: *name*, representing the name of the variable and *type*, representing its type (boolean, string, integer, etc.). The function meta-class represents any function that can act

on the smart contract by changing its status. Function can have an output parameter (represented as a Variable) and a set of input parameters (also represented as a Variable).

There are several types of Functions. The most general is Function. This corresponds to Function concept in Figure 5. It is a procedure that can consult any parameter of the smart contract, but does not change anything. Apart from the general type, there are three special types of Functions:

Constructor This is a function that creates a new instance of the smart contract. Each smart contract must have at least one of this type of Function (either implicitly or explicitly).

Modifier This is a type of function that can be used to agilely change the behavior of functions. It can automatically check a condition before executing a function.

Event This is another type of function. If an event is emitted, it stores the arguments passed in transaction logs. These logs are stored in blockchain and are accessible using the address of the contract for as long as the contract remains present in the blockchain. An event is actually a function that throws a trigger over another smart contract in the blockchain procedure.

As the names used in the meta-model are the same as the ones, we used in Figures 4 and 5, it is easy to make a simple comparison between the meta-model and real models. It is also very important to note that this meta-model only represents smart contracts. In future work we want to enrich it to incorporate concepts that will let us represent blockchain environments. It should also be mentioned that this meta-model, will constitute the baseline for the automatic testing of smart contracts, as reported in [19]

5.2 Smart Contract Profile

As stated earlier in this section, our proposal is to adopt MOF and UML principles in our smart contract approach. With this in mind, and to make our meta-model in Figure 6 compatible with UML, in this section we propose a UML profile.

It is shown in Figure 7. The structure that we used for our meta-model in Figure 6 corresponds fairly closely to UML structure. In fact, the idea of differentiating the structural and behavioural parts helped simplify our definition of profile. Also, as we had a set of classes (Modifier, Constructor and

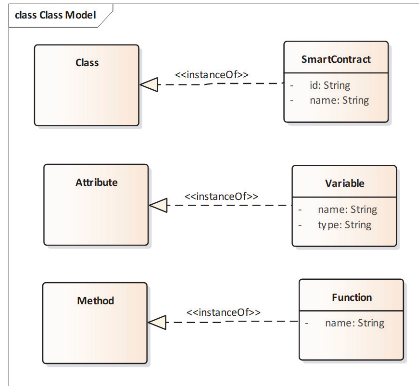

Figure 7 An UML-Profile for smart contract meta-model.

Event) inherited from Function, we were able to obtain a very compact profile with only three definitions (see Figure 7). The meta-class smart contract is an instance of UML "Class", Function is an instance of UML "Method" and Variable is an instance of UML "Attributes".

The simplicity of this profile does not mean that it is not a powerful tool. The definition of the meta-model and the profile allow as to clearly define any concept involved in a smart contract and match each concept with similar concepts in UML. We can therefore use the extension mechanisms that UML offers to obtain a formal definition of a smart contract. This will be the base for the automatic tests we intend to generate in future works.

6 Conclusions and Future Work

This paper summarizes the main concepts of blockchain, the general characteristics and anatomy of smart contracts. It also outlines the conceptualization of model-based software engineering and the need for software quality assurance of smart contracts. We also discuss the structure of different smart contracts and, given the benefits it brings, the use of model-based smart contract engineering for verification, validation and code generation is proposed. Subsequently, we analyze related works and indicate the general deficiencies that are appreciated from our point of view, and finally we present a new approach for a smart contract metamodel. Regarding the related works, indicate that none of them allows the automatic generation of test cases, which is the ultimate goal of this paper.

The idea of the formal approach presented is to try to define a framework that will help us establish an automatic process for validating smart contracts, i.e., the automatic generation of test cases. As presented in another paper [19], one of our future goals is actually to define a mechanism for automatizing smart contracts using a model-driven approach. This idea has been applied in other fields, like the that presented by Escalona et al. [28]. This work will be the basis for work to improve the combination of smart contracts and computational law by designing a legal-oriented smart contract model. We also plan to develop a MDSE tool that will automatically detect conformance between contract code and models, even in the natural language context. This will accelerate the extension and development of smart contract engineering, similar to that presented by Meidan et al. [29].

The work presented in this paper focused on smart contracts but, as has been pointed out, a smart contract is a part of a blockchain. In future work, we intend to include the concept of blockchain in our meta-model and ensure that the testing we carry out on smart contracts is in line with that overall concept.

Another possible line of activity is the transfer of our work. Previous works related to MDSE for early testing have been successful in terms of industry transference [30]. One of the main contributing factors to achieve this success has been to offer a suitable tool to interact with meta-models and transformations. In this regard, future work clearly needs to address the implementation of our profile in a tool and the implementation of our testing mechanisms in order to consolidate our results. Our idea is to design a case tool based on a UML tool like those defined in other approaches such as NDT (Navigational Development Techniques) [31].

Acknowledgments

This research was partially supported by the NICO Project (PID2019-105455GB-C31) of the Spanish Government's Ministry of Economy and Competitiveness and the Trop@ (CEI-12) Project of Regional Government of Andalusia, Spain.

References

[1] Satoshi, N. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Available online at: https://bitcoin.org/bitcoin.pdf

[2] Westerkamp, M., Victor, F., and Kupper, A. (2020). *Tracing manufacturing processes using blockchain-based token compositions*. Digital Communications and Networks, 6(2):167—176.

[3] Shailak, J. (2020). *Smart Contracts: Building Blocks for Digital Transformation*. Available online at: https://doi.org/10.13140/RG.2.2.33316.83847

[4] Viriyasitavat, W., and Hoonsopon, D. (2019). *Blockchain characteristics and consensus in modern business processes*. Journal of Industrial Information Integration, 13, 32–39.

[5] Nanayakkara, S., Rodrigo, M. N. N., Perera, S., Weerasuriya, G. T., and Hijazi, A. A. (2021). *A methodology for selection of a Blockchain platform to develop an enterprise system*. Journal of Industrial Information Integration, 23, 100215.

[6] Tsung-Ting Kuo, Hugo Zavaleta Rojas, Lucila Ohno-Machado (2019). *Comparison of blockchain platforms: a systematic review and healthcare examples*. Journal of the American Medical Informatics Association, Volume 26, Issue 5, May 2019, Pages 462—478, https://doi.org/10.1093/jamia/ocy185

[7] Huang, F. (2017). *Human Error Analysis in Software Engineering.* Available online at: https://doi.org/10.5772/intechopen.68392.

[8] Object Management Group, Inc. Available online at: https://www.omg.org/spec/UML/About-UML/

[9] Forward, A., and Lethbridge, T. (2008). *Problems and opportunities for model-centric versus code-centric software development: A survey of software professionals*. International Workshop on Models in Software Engineering. Available online at: https://doi.org/10.1007/s11408-018-0315-6

[10] Drescher, D. (2017). *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. Apress, USA.

[11] Lu, Q., Weber, I., and Staples, M. (2018). *Why Model-Driven Engineering Fits the Needs for Blockchain Application Development.* IEEE Blockchain Technical Briefs, September 2018.

[12] Wikipedia Blockchain. Available online at: https://en.wikipedia.org/wiki/Blockchain

[13] Tapscott, D. and Tapscott, A. (2016). *The Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World*. pp. 72, 83, 101, 127. ISBN 978-0670069972.

[14] Buterin V., (2014). *A next-generation smart contract and decentralized application platform*. White paper, 2014, vol. 3, no 37. Available online

at: https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf

[15] Hyperledger Project. Available online at: https://www.hyperledger.org

[16] Lewis A. (2016), *A gentle introduction to smart contracts*, Available online at: https://bitsonblocks.net/2016/02/01/a-gentle-introduction-to-smart-contracts/

[17] Mohanta, B. K., Panda, S. S., and Jena, D. (2018). *An overview of smart contract and use cases in blockchain technology.* In 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), pages 1—4. IEEE.

[18] Xu, X., Lu, Q., Liu, Y., Zhu, L., Yao, H., and Vasilakos, A. V. (2019). *Designing blockchain-based applications a case study for imported product traceability*. Future Generation Computer Systems, 92:399—406.

[19] Sanchez-Gomez, N. et al. (2020). *Current Limitations of Blockchain Traceability: Challenges from Industry*. WEBIST 2020. 16TH International Conference on Web Information Systems and Technologies.

[20] The Ethereum community. Available online at: https://ethereum.org/en/developers/docs/programming-languages/

[21] Schmidt DC. (2006), *Guest Editor's Introduction: Model-Driven Engineering*. Computer 2006 Feb;39(2):25—31.

[22] Hu, K., Zhu, J., Ding, Y., Bai, X., and Huang, J. (2020). *Smart Contract Engineering*. Electronics, 9(12), 2042.

[23] Ladleif, J., and Weske, M. (2019, November). *A unifying model of legal smart contracts*. In International Conference on Conceptual Modeling (pp. 323–337). Springer, Cham.

[24] Lu, Q., Binh Tran, A., Weber, I., O'Connor, H., Rimba, P., Xu, X., ... and Jeffery, R. (2020). *Integrated model-driven engineering of blockchain applications for business processes and asset management*. Software: Practice and Experience.

[25] Vandenbogaerde, B. (2019). *A graph-based framework for analysing the design of smart contracts.* In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (pp. 1220–1222).

[26] Butijn, B. J., van den Heuvel, W. J., and Kumara, I. (2019). *Smart Contract-Driven Business Transactions.* Essentials of Blockchain Technology, 81.

[27] Skotnica, M., Klicpera, J., and Pergl, R. (2020). *Towards Model-Driven Smart Contract Systems–Code Generation and Improving Expressivity of Smart Contract Modeling.*

[28] Escalona Cuaresma, M.J., Gutiérrez Rodríguez, J.J., Mejías Risoto, M., Aragón Serrano, G., Ramos Román, I. (2011). *An Overview on Test Generation from Functional Requirements*. The Journal of Systems and Software. Vol. 84. Núm. 8. Pag. 1379-1393.

[29] Meidan, A., García-García, J. A., Ramos, I., & Escalona, M. J. (2018). *Measuring software process: a systematic mapping study.* ACM Computing Surveys (CSUR), 51(3), 1–32.

[30] López, G., García-Borgoñón, L., Vega, S., Escalona Cuaresma, M.J., Juristo, N. (2020). *Cultivating Practitioners for Software Engineering Experiments in industry. Best Practices learned from the experience*. Pag. 1–12. Advancements in Model-Driven Architecture in Software Engineering. EEUU. IGI Global. ISBN 1799836614

[31] Escalona Cuaresma, M.J., Aragón Serrano, G. (2008). *NDT. A Model-Driven Approach for Web Requirements*. IEEE Transactions on Software Engineering. Vol. 34. Núm. 3. Pag. 377–390.

Biographies

N. Sánchez-Gómez received the degree in computer engineering and the master's degree in engineering and software technology from the University of Seville, with the knowledge and skills of people management, ICT project management, customer management and practical application of computer engineering methodologies and techniques. He is currently a Researcher with the Department of Computer Languages and Systems, ETSII. University of Seville.

Since 1990 to 2001, he worked with Coritel (Accenture group), where he also carried out management and project management activities. From 2001

to 2009, he developed his professional activity as a Manager of Everis Spain, being responsible for different accounts in both the public and private sectors. He has developed a large part of its professional career in the technology and process consultancy sector, both in the private and public sectors. Throughout more than thirty years of professional experience, he has gone from implementing ICT solutions to supervising work teams, managing clients, and leading ICT projects. He is currently a member of the Web Engineering and Early Testing Research Group. He has a broad knowledge of the functions and processes that make up the activity environment of the sectors in which he has participated.

Further information about her research activities and her list of publications can be found at https://investigacion.us.es/sisius/sis_showpub.php?idpers=20733

J. Torres-Valderrama received his MSc and the Phd in Computer Systems from Seville University. He has been working in the Department of Computer Languages and Systems at the Seville University since 1991, where he is currently a senior lecturer. Her main research interests are related to requirements engineering, web-based systems development, user interfaces, usability, and early software testing. In these areas, he has directed several PhD theses and published numerous papers in journals and congresses. He has managed and participated in a high number of projects related to her areas of research.

He has been dean of School of Computer Engineering at Seville's University from 2006 to 2014 and he is currently manager of the Foundation for Research and Development of Information Technology in Andalusia since 2016.

Further information about her research activities and her list of publications can be found at https://investigacion.us.es/sisius/sis_showpub.php?idpers=3278

Manuel Mejías Risoto received his degree in Industrial Engineering in 1985 and his PhD degree in Industrial Engineering in 1997 from the University of Seville (Spain). He has been a professor of software engineering at the University of Seville since 1987. Dr. Mejías has focused his research activity on Object Oriented System Modeling, Software Development Process, Testing, Software Quality, Metrics and Software Project Management.

Further information about her research activities and her list of publications can be found at https://investigacion.us.es/sisius/sis_showpub.php?idpers=3270

Alejandra Garrido is a full professor at Facultad de Informática, Universidad Nacional de La Plata, Argentina, where she is Associate Director of the Research and Development in Advanced IT Lab (LIFIA). She is also a researcher at CONICET. Her research interests are focused on software evolution through refactoring and the application of patterns, to improve both internal and external qualities of software. Alejandra is also the Director of the Master degree in Software Engineering. She holds a PhD in Computer Science from the University of Illinois at Urbana-Champaign (UIUC). She is also a member of the Hillside Group.