SCMTool: A Graphical Tool for Smart Contract Modeling

Gislainy Crisostomo Velasco[1], **Dionatan Alves Vieira**[1], **Marcos Alves Vieira**[1,2], **Sergio T. Carvalho**[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74001-970 – Goiânia – GO – Brasil

[2]Instituto Federal de Educação, Ciência e Tecnologia Goiano (IF Goiano)
76200-000 – Iporá – GO – Brasil

`gislainycrisostomo@discente.ufg.br`, `dionatan03_alves@discente.ufg.br`,

`marcos.vieira@ifgoiano.edu.br`, `sergio@inf.ufg.br`

Abstract. *The development of smart contracts in the Ethereum Virtual Machine (EVM) can be a complex task, both for experienced and beginner developers. Understanding these contracts can be challenging for both technical and non-technical users, due to the difficulty in comprehending the connection between the elements and resources available, as there is no clear way to visually present the functionalities of a contract and its relationships. In this paper, we introduce the Smart Contract Modeling Tool (SCMTool), a graphical tool based on the Model-Driven Engineering approach, that allows users to specify models that represent the structure of a smart contract in a simpler and more intuitive way. The tool was validated using a use case from the NFT industry.*

1. Introduction

Blockchain technology has enabled the implementation of smart contracts. However, the immutability of blockchain technology has significant implications for the development of Decentralized Applications (DApps), requiring software developers to implement a new programming approach. Once the source code of a contract is implemented, it cannot be modified, which demands a more rigorous development process. As a result, contract developers are faced with a series of challenges, making the activity complex and error-prone, for both experienced and novice developers [Jurgelaitis et al. 2022, Ferreira et al. 2020]. This represents a barrier to the formation of new developers in the field due to the high learning curve [Garamvölgyi et al. 2018, Jiao et al. 2020]. For non-technical users, understanding contracts can be challenging due to the difficulty in understanding the connection between elements and available resources, as there is no clear way to visually present a contract's functionalities and relationships [Qasse et al. 2021].

To minimize human errors in creating contracts, several authors have suggested the use of software engineering techniques, including Model-Driven Engineering (MDE) [Ben Slama Souei et al. 2021, Chirtoaca et al. 2020, Guida and Daniel 2019, Hamdaqa et al. 2020, Santiago et al. 2021]. One of the techniques of MDE is the construction of graphical models that help stakeholders to better understand the problem domain [Viyović et al. 2014]. Therefore, this paper aims to propose the development of a graphical tool for the development of smart contracts that enables the modeling of the smart contract through a graphical interface.

The rest of this paper is organized as follows: in Section 2, we present the main concepts covered in this paper. In Section 3, we present related works and compare them with this work. Then, in Section 4, we present the SCMTool and its main components. In Section 5, we present a use case from the NFT industry. Finally, in Secton 6, we present the final considerations and future work.

2. Background

Smart contracts are a set of programming instructions used to establish agreements in a decentralized and reliable manner on blockchains [Wang et al. 2019]. The Ethereum Virtual Machine (EVM) is a platform for executing smart contracts that provides a secure and reliable environment for the execution of these contracts [Buterin et al. 2014]. In addition to the EVM, other blockchain platforms such as Hyperledger Besu, EOS, Tron, Cardano, and Polygon are compatible with the EVM, allowing contracts to be written on one platform and reused on others, which facilitates the development of DApps.

Model-Driven Engineering (MDE) is a methodology that uses models as primary artifacts to represent requirements, behaviors, and functionalities of the system [Seidewitz 2003]. With this approach, technical and non-technical professionals can share knowledge and communicate more efficiently during software development. MDE makes use of models, which are abstractions of the system under study, and employs techniques and tools to transform models into executable code, including code generation, validation, and model transformations [Seidewitz 2003, Iovino et al. 2012]. Graphical tools can also be used to assist in the modeling of the system, allowing users to create models visually without the need for advanced programming knowledge.

3. Related Works

There have been several studies aimed at enhancing the field of smart contract development using various approaches, such as Model-Driven Architecture (MDA), MDE, Unified Modeling Language (UML) diagrams, Business Process Model and Notation (BPMN), and others. Of these approaches, BPMN stands out due to its graphical nature, which enables clear and visual modeling of business processes using symbols and diagrams.

On the other hand, some studies utilizing different techniques require the development of an additional graphical tool. For example, the work [Hamdaqa et al. 2020] proposes a metamodel called IContractML for smart contract modeling, along with a graphical tool for manipulation that utilizes Sirius[1]. However, due to the limitations of the adopted reference model, the metamodel may not fully capture all aspects of smart contract construction. In contrast, we propose a tool that utilizes a high-level metamodel for the construction of smart contracts for the EVM. Additionally, in [Boubeta-Puig et al. 2021], the authors present a graphical editor for event-driven smart contracts that uses the Graphical Modeling Framework (GMF). However, we choose to use Sirius, a more declarative tool.

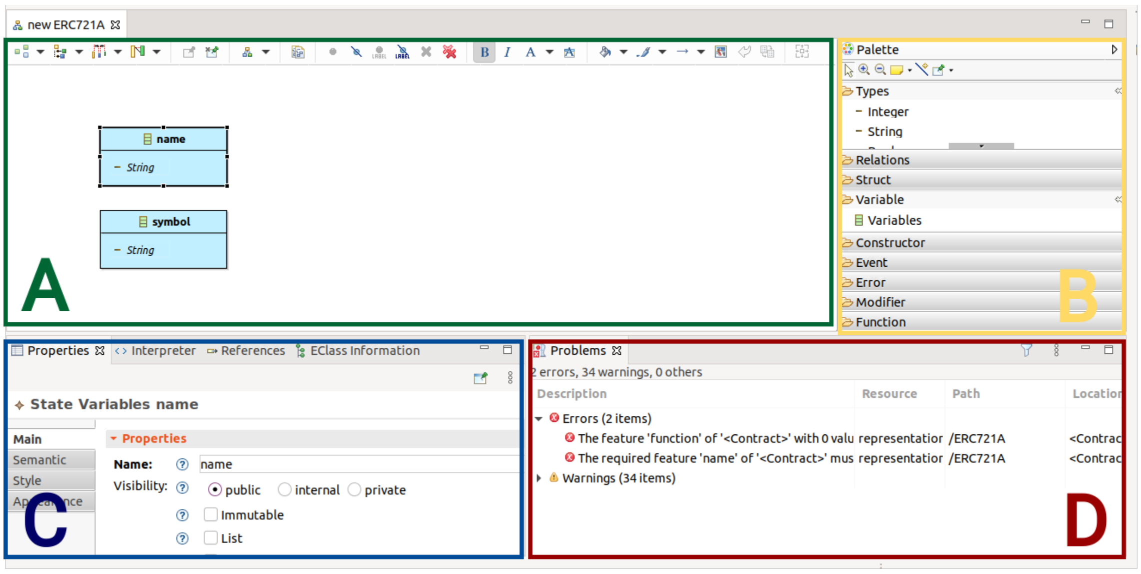
Block-based programming also has been used in previous studies to enable the construction of a visual tool where users drag predefined blocks to create smart contracts.

[1]Sirius is a complete framework that provides resources for developing a fully functional graphical modeling environment.

For example, the authors of [Guida and Daniel 2019] present a model for contract development that encompasses the concepts of service orientation. The study utilized other tools, such as Doxity, which generates website documentation from Solidity source code. Moreover, the work [Merlec et al. 2021] employs block programming to build a contract creation tool for Hyperledger Fabric. Both studies aim to provide a more user-friendly solution, which is also the motivation of this paper. However, they use different approaches to achieve this objective.

4. Smart Contract Modeling Tool (SCMTool)

SCMTool utilizes the High-Level Metamodel for Smart Contract (HLM-SC), a metamodel that allows for high-level declaration of smart contract elements [Velasco and Carvalho 2022b]. HLM-SC provides a set of components that enable the representation of smart contracts, including state variables, complex data structures, functions, modifiers, events, errors, and a constructor. These components are represented in the tool by rectangular shapes, and their relationships are depicted through edges. Figure 1 illustrates SCMTool and its main components.



State Variables name

Main
Semantic
Style
Appearance

Properties
Name: name
Visibility: [✓ public] internal private
Immutable
List

2 errors, 34 warnings, 0 others

Description	Resource	Path	Location
Errors (2 items)			
The feature 'function' of '<Contract>' with 0 valu	representatior	/ERC721A	<Contrac
The required feature 'name' of '<Contract>' mus	representatior	/ERC721A	<Contrac
Warnings (34 items)			

Figura 1. SCMTool

The contract is modeled in the main area (A), where the user can drag and drop the available components from the palette (B). The palette contains all the components that make up the contract, and when a rectangular shape is selected, its properties can be modified (C). If the modeled contract does not conform to the specifications of the HLM-SC, errors are displayed in this area (D). The SCMTool has validation rules based on the specifications of the HLM-SC, so a component can only be dragged onto a rectangular shape when there is a match with the HLM-SC specification.

Figure 2 illustrates the use of these graphical components. State variables are represented by rectangular shapes, such as *_certificate* (blue color). The variable *_certificate* uses the complex data structure represented by *Certificate* (green color). The shape labeled *mint* represents a function (white color) and uses the modifier *onlyAuthorizedAddress* (orange color). This modifier throws an error *NotAuthorized* if the calling address is

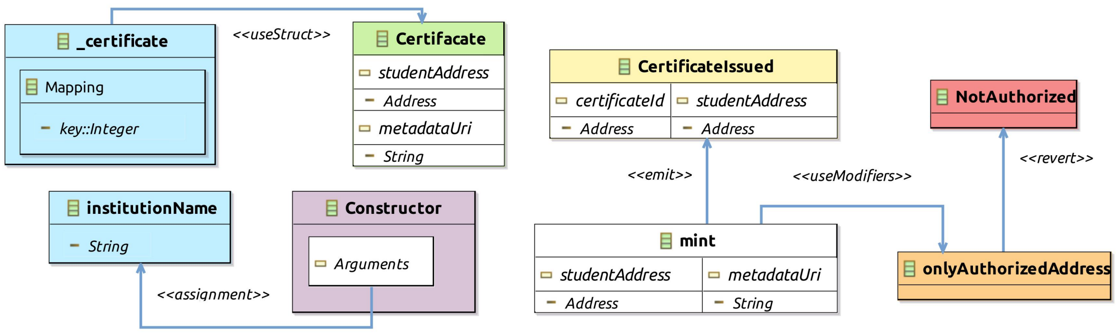


Figura 2. The main graphical components and their relationships.

unauthorized (red color). In case of success, the function *mint* emits the event represented by the shape *CertificateIssued* (yellow color). Finally, the constructor (purple color) initializes the state variable *institutionName*.

5. Use Case: NFT Contract

The tool validation was performed by modeling an NFT contract. NFTs are unique and non-fungible tokens that can represent both physical and digital items [Velasco and Carvalho 2022a, Wang et al. 2021]. In a Solidity contract, NFT tokens are represented by a structure similar to a hash table, where each token has only one owner, and all transfers of its ownership are recorded on the blockchain through the emission of an event, enabling its traceability through DApps.

The modeling followed the specifications of the ERC-721 standard[2], which defines the mandatory elements in an NFT contract. In this case, we used the implementation of the ERC721A contract[3], which allows for batch issuance of multiple NFTs in a single transaction with an optimized operation fee. Figure 3 illustrates the final version of the modeled contract[4].

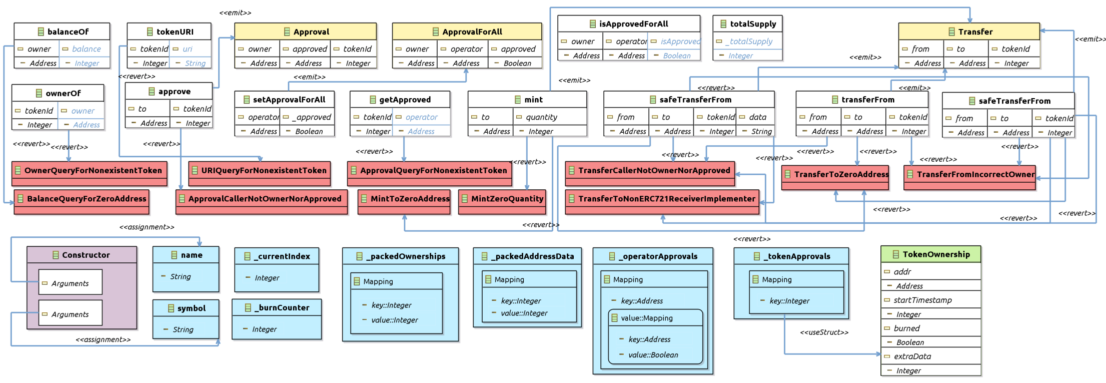


Figura 3. NFT Contract modeled in SCMTool.

[2]https://eips.ethereum.org/EIPS/eip-721

[3]https://github.com/chiru-labs/ERC721A

[4]For space reasons, the source code of the modeled contract and additional information about the tool are available for consultation at https://github.com/gislainy/scmtool-short-paper.

The contract described in the tool has eight state variables, four of which are mappings highlighted in blue color. The *_tokenApprovals* variable uses the *TokenOwnership* structure, which is represented in green color. The constructor initializes the *name* and *symbol* variables, highlighted in purple color. Among the twelve implemented functions, identified in white color, six of them are view functions, while the other six modification functions always emit corresponding events, such as the *setApprovalForAll* function that emits the *ApprovalForAll* event, highlighted in yellow color. An event can be used by more than one function, as is the case with the *Transfer* event used in both the *mint* function and the other transfer functions. Eleven types of errors were modeled in the contract, highlighted in red color, which are reused in several functions.

6. Conclusion

In this paper, we presented a graphical tool that assists both technical and non-technical individuals in visualizing the elements of a smart contract clearly. This enables more efficient communication among stakeholders and helps to minimize human errors. Although the tool presents a high-level specification of smart contract elements, the language used to represent these elements can be improved with other forms of representation or even with the creation of a graphical language. In addition to this improvement, our next steps include conducting an evaluation to assess the quality of the tool.

Referências

Ben Slama Souei, W., El Hog, C., Sliman, L., Ben Djemaa, R., and Ben Amor, I. A. (2021). Towards a uniform description language for smart contract. In *2021 IEEE 30th WETICE*, pages 57–62.

Boubeta-Puig, J., Rosa-Bilbao, J., and Mendling, J. (2021). Cepchain: A graphical model-driven solution for integrating complex event processing and blockchain. *Expert Systems with Applications*, 184:115578.

Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform. *white paper*, 3(37):2–1.

Chirtoaca, D., Ellul, J., and Azzopardi, G. (2020). A framework for creating deployable smart contracts for non-fungible tokens on the ethereum blockchain. In *2020 IEEE DAPPS*, pages 100–105.

Ferreira, J. F., Cruz, P., Durieux, T., and Abreu, R. (2020). Smartbugs: A framework to analyze solidity smart contracts. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1349–1352.

Garamvölgyi, P., Kocsis, I., Gehl, B., and Klenik, A. (2018). Towards model-driven engineering of smart contracts for cyber-physical systems. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 134–139.

Guida, L. and Daniel, F. (2019). Supporting reuse of smart contracts through service orientation and assisted development. In *2019 DAPPCON*, pages 59–68.

Hamdaqa, M., Metz, L. A. P., and Qasse, I. (2020). 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. Association for Computing Machinery.

Iovino, L., Pierantonio, A., and Malavolta, I. (2012). On the impact significance of metamodel evolution in mde. *J. Object Technol.*, 11(3):3–1.

Jiao, J., Lin, S.-W., and Sun, J. (2020). A generalized formal semantic framework for smart contracts. In *Fundamental Approaches to Software Engineering: 23rd International Conference, FASE 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25–30, 2020, Proceedings*, page 75–96, Berlin, Heidelberg. Springer-Verlag.

Jurgelaitis, M., čeponienė, L., and Butkienė, R. (2022). Solidity code generation from uml state machines in model-driven smart contract development. *IEEE Access*, 10:33465–33481.

Merlec, M. M., Lee, Y. K., and In, H. P. (2021). Smartbuilder: A block-based visual programming framework for smart contract development. In *2021 IEEE Blockchain*, pages 90–94.

Qasse, I., Mishra, S., and Hamdaqa, M. (2021). icontractbot: A chatbot for smart contracts' specification and code generation. In *2021 IEEE/ACM 3rd BotSE*, pages 35–38.

Santiago, L., Abijaude, J., and Greve, F. (2021). A framework to generate smart contracts on the fly. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering*, SBES '21, page 410–415, New York, NY, USA. Association for Computing Machinery.

Seidewitz, E. (2003). What models mean. *IEEE Software*, 20(5):26–32.

Velasco, G. and Carvalho, S. (2022a). Domínios, aplicações, desafios e oportunidades sobre non-fungible tokens (nft): Um mapeamento sistemático da literatura. In *Anais do VII WASHES*, pages 41–50, Porto Alegre, RS, Brasil. SBC.

Velasco, G. and Carvalho, S. (2022b). Uma abordagem dirigida por modelo para desenvolvimento de contratos inteligentes na ethereum virtual machine. In *Anais da X Escola Regional de Informática de Goiás*, pages 106–117, Porto Alegre, RS, Brasil. SBC.

Viyović, V., Maksimović, M., and Perisić, B. (2014). Sirius: A rapid development of dsm graphical editor. In *IEEE 18th INES 2014*, pages 233–238.

Wang, Q., Li, R., Wang, Q., and Chen, S. (2021). Non-fungible token (nft): Overview, evaluation, opportunities and challenges. *arXiv preprint arXiv:2105.07447*.

Wang, S., Ouyang, L., Yuan, Y., Ni, X., Han, X., and Wang, F.-Y. (2019). Blockchain-enabled smart contracts: Architecture, applications, and future trends. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(11):2266–2277.