Model-driven approach for the design of multi-chain smart contracts

Ankica Barisic, Enlin Zhu, Frédéric Mallet

▶ **To cite this version:**

Ankica Barisic, Enlin Zhu, Frédéric Mallet. Model-driven approach for the design of multi-chain smart contracts. 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), Sep 2021, Paris, France. 10.1109/BRAINS52497.2021.9569809 . hal-03338936

HAL Id: hal-03338936
https://hal.science/hal-03338936

Submitted on 9 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Model-driven approach for the design of multi-chain smart contracts

Ankica Barišić*, Enlin Zhu* and Frédéric Mallet*
* Université Côte d'Azur, CNRS, Inria, I3S, 06900 Sophia Antipolis, France

***Abstract*—Blockchain-based smart contracts provide transparent automation in a broad range of services, including finance, the Internet of Things, and autonomous systems. However, the implementation of such services may easily involve security risks and functional errors, especially for complex services composed of different blockchains. To help developers focus on their business model instead of diving into the blockchain architecture heterogeneity, we propose a framework to enable analysis and comparison of composed services before deployment. This is achieved through the intensive use of model-based engineering allowing reasoning on the model before generating concrete deployment artifacts, especially for the safe orchestration of contract calling transactions.**

***Index Terms*—multichain, smart contract, model-driven engineering, formal verification**

I. Introduction

As a piece of executable code residing at a specific address on a distributed ledger [1], the blockchain-based smart contract permits irreversible executions of transparent transactions among entities with distrust. Taking the IoT scenario as an example, smart contracts can be used to automate data sharing, device ownership transfer, user registration, certificate deployment and revocation, etc. Smart contracts also enable the implementation of cryptocurrency as an incentive, which helps create a digital market to trade resources across multiple systems, such as smart grids, transport systems, and logistic networks.

The whole ecosystem includes stakeholders with different behavior guarantees. For the sake of the trade-off between the fault-tolerance capacity and the transaction speed, developers are encouraged to use different blockchains to address corresponding stakeholders. However, the outcome of their interactions is not as clearly defined as the transactions on one single chain. To provide additional guarantee to the composed service, Model Driven Engineering (MDE) is used. MDE provides explicit models, commonly called 'first-class artifacts', that are further translated to lower level models with more details. This approach is adopted to large-scale software engineering problems thanks to its rapid prototyping, simulation, validation and verification capabilities. Such models are expressed through different Domain-Specific Modeling Languages (DSMLs) [2].

MDE enables reasoning on the model before generating concrete artefacts for deployment, including the orchestration of services and the specification of blockchain technologies. By having appropriate DSMLs we can contribute to the creation of a service-level semantics of contracts running on given multi-chain back-ends, based on which some security and temporal properties could be formally verified by enabling simulation at appropriate levels of abstraction.

978-1-5386-4235-1/18/$31.00 ©2018 IEEE

II. Related Work

In single-chain scenarios, business model translation and smart contract verification have been explored separately. Taking financial application as an example, the Digital Asset Modeling Language (DAML)[1] is used in asset management logic description by offering a solution at a higher level of abstraction. However, this approach lacks the integration of verification features. On the verification side, Tezos blockchain [3] has already considered formal verification problems during the protocol design phase and code implementation. Properties are extracted from virtual machine code and fed to a theorem prover like Coq [4]. For other blockchain projects without such a feature, additional formalization frameworks like the K-framework [5] and state machine abstractions give the formal semantics.

There are research efforts devoted to generating Ethereum-based smart contracts using MDE [6], even by generating code from natural language models [7]. *FSolidM* is such a solution that provides formal definitions to address vulnerabilities like reentrancy and transaction ordering, as well as to support design patterns like time constraints and authorizations [8]. It, therefore, gives the developers a new way to formulate their smart contracts by providing support for verification, but the business model is still tight to Solidity implementation.

On multi-chain scenarios, several frameworks of cross-blockchain applications have been proposed, but their communication cost is not explicit [9]. BlockSY [2] proposes a technology-agnostic web services to ease changing the underlying blockchain. While it deals with the syntactic adaptation, it does not explicitly define the expected semantics of transactions and tightly relies on the one of underlying blockchains.

III. Development of multi-chain Smart Contract

We assume that the entire business logic is based on a working flow. Each step of this working flow has a set of implementations in different blockchains in the form of libraries. The current cross-blockchain interaction model is simple message passing without state synchronization.

[1]https://daml.com/ (Accessed: 12.04.2021)
[2]https://blocksy-wiki.symag.com/ (Accessed: 12.04.2021)

The objective of our approach is to wisely reuse existing libraries implemented in different blockchains and explicitly specify and analyze the behavior of the generated composed smart contracts. We need to orchestrate steps on different blockchains while ensuring that generated smart contracts satisfy expected properties. When several valid orchestrations are possible, we choose the one that optimizes a non-functional property, like the execution time or the gas consumption.

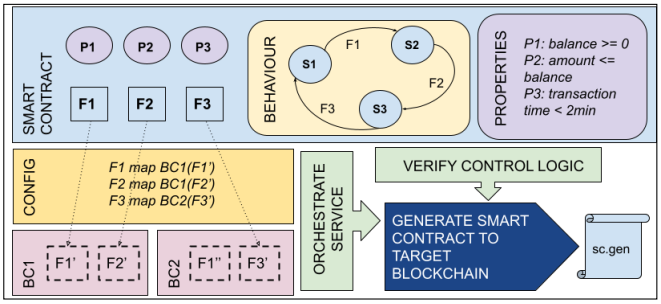

Fig. 1. MDD approach for generating multi-chain smart contracts

Our approach is expressed by a three-layer framework with a synthesis phase. The top layer is about the service design, including three elements: the working flow of the business model, the specification of each step, and the specification of the business model. The bottom layer is a set of candidate blockchain platforms that host the libraries to implement each step of the working flow above. The middle layer is a configuration that maps each step to a corresponding function in a chosen library on one blockchain platform. We use three DSMLs to separately describe the abstract behavior (working flow), the properties (specification of the business model) and the configuration (step-function mapping). The whole service is based on a smart contract hosted by a chosen blockchain which invokes functions residing in libraries deployed on other blockchains. This smart contract is expected to be generated from the abstract behavior, the configuration and an accepted cross-blockchain context parsing rule.

Assuming that the chosen functions in the configuration are justified by the single chain verification methods, once the control logic of the working flow is formally justified, the generated artefacts of the whole service will satisfy the business model specification.

Our approach relies on the GeMoC studio [10], an Eclipse-based language and modeling workbench. The framework provides a generic interface to bring together multiple DSMLs and describe the semantics of their coordination. It allows plugging-in different execution engines for each of their specific metalanguages. We have developed several concrete syntaxes to specify model instances for smart contracts, libraries, and deployed configurations.

To analyze the synthesized service behavior, we can use the GeMoC tools for simulation, model checking, debugging, and run-time verification. GeMoC also has a support to generate a full semantic model from a set of patterns applied to each DSML. This semantic model is described using the Clock Constraint Specification Language (CCSL) that provides a formal support for conducting exhaustive verification[11].

Our approach opens the possibility to introduce new functionalities on top of the smart contracts, which can be defined by third parties. One possible application is the addition of the specific language for defining the legal clauses which can be used to provide governance over the smart contract while in runtime. We can, for instance, introduce concepts like termination of contract or adding addendum to the existing clauses of contracts. On the other hand, having the contract behavior explicitly defined, it is then possible to provide a simulation environment for monitoring the contracts at run-time.

IV. Conclusion

This work provides a framework to address the reliability problem of multi-chain smart contract design. Although the code generation part is not finished, we build some automatic supports to describe the behavior of contracts and orchestrate the execution of transactions. The main contribution is the creation of the semantics of the service composition of a multi-chain interaction model using the model-driven approach, which could be transferred into existing verification tools.

References

[1] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger. tech- nical report eip-150, ethereum project - yellow paper."

[2] J. Gray, J.-P. Tolvanen, S. Kelly, A. Gokhale, S. Neema, and J. Sprinkle, "Domain-specific modeling," *Handbook of Dynamic System Modeling*, pp. 1–7, 2007.

[3] B. Bernardo, R. Cauderlier, Z. Hu, B. Pesin, and J. Tesson, "Mi-Cho-Coq, a framework for certifying Tezos Smart Contracts," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12232 LNCS, pp. 368–379, 9 2019. [Online]. Available: http://arxiv.org/abs/1909.08671

[4] Y. Bertot and P. Castéran, *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[5] G. Roşu and T. F. Şerbănuţă, "An overview of the K semantic framework," *Journal of Logic and Algebraic Programming*, vol. 79, no. 6, pp. 397–434, 8 2010.

[6] A. Mülder, "Model-Driven Smart Contract Development for Everyone," *Hacker Noon*, 2019. [Online]. Available: https://hackernoon.com/model-driven-smart-contract-development-for-everyone-jiu32p0

[7] O. Choudhury, N. Rudolph, I. Sylla, N. Fairoza, and A. Das, "Auto-Generation of Smart Contracts from Domain-Specific Ontologies and Semantic Rules," *Proceedings - IEEE 2018 International Congress on Cybermatics: 2018 IEEE Conferences on Internet of Things, Green Computing and Communications, Cyber, Physical and Social Computing, Smart Data, Blockchain, Computer and Information Technology, iThings/Gree*, pp. 963–970, 2018.

[8] A. Mavridou and A. Laszka, "Designing Secure Ethereum Smart Contracts: A Finite State Machine Based Approach," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10957 LNCS, no. November 2017, pp. 523–540, 2018.

[9] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia, "A survey on blockchain interoperability: Past, present, and future trends," *arXiv*, 2020.

[10] E. Bousse, T. Degueule, D. Vojtisek, T. Mayerhofer, J. Deantoni, and B. Combemale, "Execution Framework of the GEMOC Studio (Tool Demo)," in *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, ser. SLE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 84–89. [Online]. Available: https://doi.org/10.1145/2997364.2997384

[11] F. Mallet and R. De Simone, "Correctness issues on MARTE/CCSL constraints," *Science of Computer Programming*, vol. 106, pp. 78–92, 2015.