Formalization and Verification of Smart Contracts Using DCR Graphs

Mojtaba Eshghie[1], Wolfgang Ahrendt[2], Cyrille Artho[1], Thomas Troels Hildebrandt[3], and Gerardo Schneider[4]

[1] KTH Royal Institute of Technology, Stockholm, Sweden
eshghie@kth.se, artho@kth.se
[2] Chalmers University of Technology, Gothenburg, Sweden
ahrendt@chalmers.se
[3] University of Copenhagen, Denmark hilde@di.ku.dk
[4] University of Gothenburg, Sweden gerardo.schneider@gu.se

1 Introduction

A *smart contract* is a reactive program that is deployed as the immutable code section of an account on a distributed ledger (e.g. blockchain). Smart contracts are used to implement a contractual agreement between multiple parties. Therefore, they are akin to special business processes specifying contractual agreements on actions to be carried out by different roles [6, 9, 14]. These contracts offer advantages such as uncompromised (automated) execution even without a trusted party. However, they can also be complex and difficult to design and understand, a problem exacerbated by the fact that the code section of smart contracts cannot be modified once deployed.

Like regular software, smart contracts embody common best practices as *design patterns* [11]. In a normal business process environment, different roles collaborate to achieve a common business goal. In contrast, different roles in a smart contract typically have adversarial interests. Therefore, smart contracts introduce new types of patterns of behavior, which have so far only been informally described [7,10,15,16]. To provide an unambiguous understanding of the patterns that can also provide the basis for formal specifications, we set out to extend the study and formalization of process patterns to include these smart contract patterns.

We use DCR graphs [12,13] capture and formalize the mentioned patterns. DCR graphs is a established declarative business process notation that has been extended with data [12], time [12], and sub-processes [13]. DCR graphs visually capture important properties such as the partial ordering of events, roles of contract users, and temporal function attributes in smart contracts. Using DCR graphs, it is possible to represent a smart contract with a clear and concise model that is more expressive and comprehensive than other types of models.

Our high-level design patterns demonstrate how a system should *behave* rather than be *implemented.* The behavior is described as restrictions on interface-level of smart contracts. It is therefore possible to use our designs as a platform- and language-independent modeling before the contract development.

Our models tell what the users interacting with the smart contract can/cannot do. Activities in our models match the transactional semantics of blockchain as an activity execution in DCR model can be mapped to a successful/mined transaction in blockchain.

Further, DCR models are useful for analysis. We show that using DCR graphs facilitates the development of correct and reliable smart contracts by providing a clear and easy-to-understand specification. This specification can be used to *monitor* the smart contract interactions both during and post contract deployment. We demonstrate that our tool HighGuard [8] can detect and flag malicious interactions given the DCR model of the contract.

2 Smart Contract Design Patterns Formalization

We systematically identify and distinguish 15 high-level design patterns from low-level (implementation-specific) patterns in smart contracts. We classify these patterns into four different categories in Fig. 1.

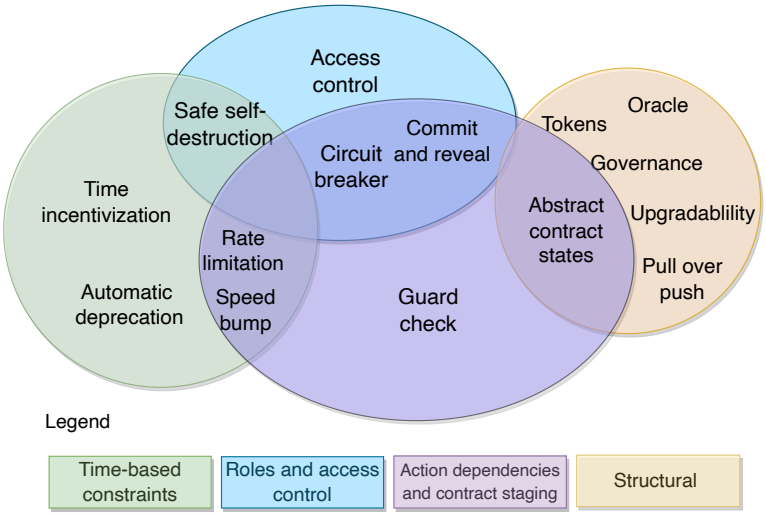

Fig. 1: Classification of smart contract high-level design patterns.

We demonstrate how one can capture the design of a full smart contract by modeling three complete contracts, not just a design pattern, with the help of DCR graphs. The modeled contracts use in total six of the design pattern models from this paper, which helps to demonstrate the combinability of pattern models to shape the final design of the contract.

3 Runtime Monitoring Using DCR Model

We present HighGuard [8] (Fig. 2), a runtime monitoring framework that leverages DCR graphs to provide runtime verification of smart contract execution. By harnessing runtime information, runtime verification techniques typically achieve significantly less false alarms compared to static analysis techniques. While the performance overhead of runtime verification is a drawback especially in blockchain ecosystem, where runtime is expensive, we mitigate this concern by executing the monitor *off-chain*. Nevertheless, HighGuard is an *online* monitor, as it observes the transactions as they are appended to the blockchain in near real-time. It persistently monitors the Ethereum network, capturing events emitted by the contract and verifying their adherence to the contract's DCR specification. Should any deviation from the specification be detected, HighGuard generates an alert.

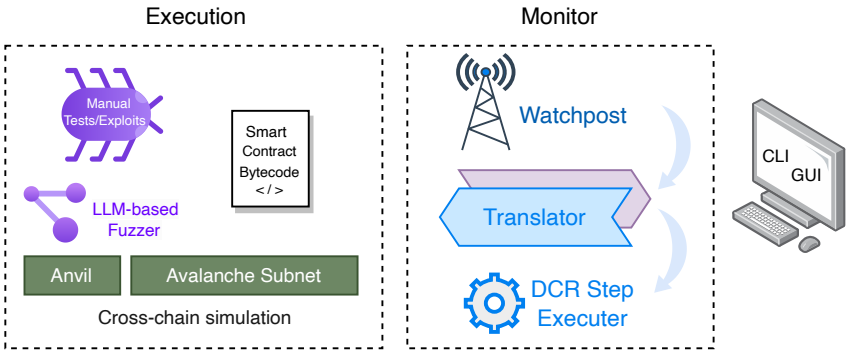

Fig. 2: HighGuard system architecture

Our usage of DCR graphs to model smart contracts and our focus on high-level rather than low-level properties allows us to capture the key semantics of the contract succinctly. We verify properties (and likewise lack of vulnerabilities pertaining to these properties) related to roles and access control [1,2], partial ordering of actions (function calls and transaction execution) [3], as well as time-based vulnerabilities [4,5].

Furthermore, not being concerned with low-level patterns and properties lets our approach remain cross-platform and not tied to the features and limitations of a certain smart contract execution environment.

References

1. SWC-105 - Smart Contract Weakness Classification (SWC), https://swcregistry.io/docs/SWC-105/, accessed: 2023-09-01

2. SWC-106 - Smart Contract Weakness Classification (SWC), https://swcregistry.io/docs/SWC-106/, accessed: 2023-09-01
3. SWC-114 - Smart Contract Weakness Classification (SWC), https://swcregistry.io/docs/SWC-114/, accessed: 2023-09-01
4. SWC-116 - Smart Contract Weakness Classification (SWC), https://swcregistry.io/docs/SWC-116/#time_locksol, accessed: 2023-09-01
5. Timestamp Dependence - Ethereum Smart Contract Best Practices, https://consensys.github.io/smart-contract-best-practices/development-recommendations/solidity-specific/timestamp-dependence/#avoid-using-blocknumber-as-a-timestamp, accessed: 2023-09-01
6. Ethereum Yellow Paper (Dec 2022), https://github.com/ethereum/yellowpaper, accessed: 2023-08-29
7. Bartoletti, M., Pompianu, L.: An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns. In: Financial Cryptography and Data Security. pp. 494–509. LNCS, Springer, Cham (2017)
8. Eshghie, M.: Mojtaba-eshghie/HighGuard (Feb 2024), https://github.com/mojtaba-eshghie/HighGuard
9. Eshghie, M., Ahrendt, W., Artho, C., Hildebrandt, T.T., Schneider, G.: Capturing Smart Contract Design with DCR Graphs. In: Ferreira, C., Willemse, T.A.C. (eds.) Software Engineering and Formal Methods. pp. 106–125. Springer Nature Switzerland, Cham (2023). https://doi.org/10.1007/978-3-031-47115-5_7
10. Fravoll: Solidity Patterns (Aug 2023), https://fravoll.github.io/solidity-patterns/, accessed: 2023-08-29
11. Gamma, E., Helm, R., Johnson, R., Johnson, R.E., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Pearson Deutschland GmbH (1995)
12. Hildebrandt, T.T., Normann, H., Marquard, M., Debois, S., Slaats, T.: Decision modelling in timed dynamic condition response graphs with data. In: Business Process Management Workshops. pp. 362–374. Springer, Cham (2022)
13. Normann, H., Debois, S., Slaats, T., Hildebrandt, T.T.: Zoom and enhance: Action refinement via subprocesses in timed declarative processes. In: BPM 2021. pp. 161–178. Springer, Cham (2021)
14. Szabo, N.: Smart contracts: building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought,(16) **18**(2), 28 (1996)
15. Wohrer, M., Zdun, U.: Smart contracts: security patterns in the Ethereum ecosystem and Solidity. In: IEEE IWBOSE. pp. 2–8 (2018)
16. Wöhrer, M., Zdun, U.: Design Patterns for Smart Contracts in the Ethereum Ecosystem. In: iThings/GreenCom/CPSCom/SmartData. pp. 1513–1520 (2018)