2019 IEEE International Conference on Software Architecture Companion (ICSA-C)

Towards Blockchain Tactics: Building Hybrid Decentralized Software Architectures

Florian Wessling, Christopher Ehmke, Ole Meyer, Volker Gruhn
University of Duisburg-Essen, Germany
Schuetzenbahn 70, 45127, Essen, Germany
firstname.lastname@uni-due.de

***Abstract*—Blockchain-based applications usually consist of centralized elements (e.g., web servers and back-end logic) connected to decentralized elements such as smart contracts. The engineering of such hybrid software architectures poses a challenge as it is unclear which elements should be centralized or decentralized. Furthermore the impact of this decision (or the balance between those two areas) on software quality attributes such as security, maintainability, performance or costs is currently unknown. The goal is to build a software architecture using the benefits and handling the challenges of blockchain technology while fulfilling the relevant quality attributes. While there are several approaches examining the relation between architectural decisions and quality attributes in centralized systems, research is at an early stage for decentralized elements in software architectures. This paper presents a first step towards architectural blockchain tactics. With a simplified experiment comparing two implementation variants of an Ethereum smart contract we show that software design patterns are not always beneficial and that the expected usage scenarios have a strong impact on the operational costs. We argue that further research and validation is necessary for gaining more qualitative and quantitative insights to make informed architectural design decisions when using blockchain technology and give a first outline on how to achieve this.**

I. Introduction

In the last few years blockchain technology emerged as a new trend to build applications based on decentralized components [1]. Similar to common information systems the main focus is on saving and retrieving data, but following the goal to decrease the central role of, e.g., a server by implementing the information processing and storage based on decentralized components. When using blockchain technology, data is stored redundantly in several locations in a data structure known as *distributed ledger*. Changes to the distributed ledger are executed in the form of transactions and kept consistent between all network participants using a consensus algorithm. Instead of trusting a central component for these systems, the trust is put in a peer-to-peer protocol based on cryptographic primitives which is kept running by a game theoretical incentive scheme. Blockchain is a so-called *Distributed Ledger Technology (DLT)*, well-known for projects such as Bitcoin [2] as a cryptocurrency and Ethereum for implementing decentralized applications [3], [4], [5].

Building blockchain-based applications enables the creation of software that is executed in a decentralized, trustless, transparent and tamper-proof environment [5] (thus often called DApps: decentralized applications). However, those advantages do not come for free. DApps, e.g., built on Ethereum, are deployed on the blockchain as *Smart Contracts (SC)*. Each interaction with an application is done through a function call on a SC that is sent as a transaction to the blockchain network. To have a transaction executed, a certain amount of "gas" (measured in Ether as the currency of Ethereum) has to be paid. The concept of gas and the requirement to specify a gas consumption limit per transaction is used to prevent denial-of-service attacks and to ensure even code with an infinite loop is stopped deterministically after a certain amount of steps [3]. These costs consist of a fixed gas amount per transaction and a variable amount of gas for the execution [4]. Execution costs depend on the complexity of the function that is called and is determined based on the computational instructions the code requires, its data structures, write-access to variables, the control flow of the SCs, etc.

Designing hybrid software architectures using blockchain technology requires a deliberate balance between centralized and decentralized elements [6]. There is a lot of interest for adding blockchain technology to an existing system or replacing existing modules with a decentralized counterpart. Nonetheless, changing an existing system is a challenging task. While blockchain technology is an opportunity for introducing decentralized elements, it is still a new concept and currently there is little common knowledge about the impact design decisions in this space have on software quality attributes such as security, maintainability, performance or costs. We therefore argue that a new blockchain-oriented view is required for the architectural design process and propose the idea of *blockchain tactics*. When it is intended to reduce the centrality of an existing system, to lower the required trust for using certain components or to add blockchain technology to a system, architectural blockchain tactics serve as a guideline. The goal is to collect approaches and best practices providing a structured way for changing a system, e.g., deciding which elements should be decentralized in order to benefit from using blockchain technology and at the same time handle its challenges. In this paper we propose a first version of *blockchain tactics* to consider the architectural design and implementation level when introducing decentralized elements. With a simplified experiment we give a first hint on the impact implementation decisions (e.g. use of design patterns) and the expected usage scenario (i.e., the frequency certain functions are going to be executed) can have on transaction costs and therefore the operational costs of the whole application.

978-1-7281-1876-5/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSA-C.2019.00048

The paper is structured as follows. In Section II we briefly discuss related approaches for designing and implementing blockchain-based software architectures. Section III explains the proposed blockchain tactics as the main contribution of this paper. Section IV presents a simplified experiment to show the impact of implementation decisions on transaction costs. In Section V we conclude our paper and discuss future work.

II. Related Work

A. Architectural Design

Xu et al. [7] present a taxonomy of blockchain properties and a flowchart with multiple questions. Together both elements help to decide which blockchain configuration is suitable for a given use case by discussing aspects like authority, storage and decentralization. Their approach refers to infrastructure and protocol options such as consensus algorithm and block size but gives no hint for the software architecture level. Wessling et al. [6] motivate why a more fine-grained approach is necessary when deciding for which elements it makes sense to be replaced or extended with blockchain technology. Their approach takes participants, their trust relations and interactions into consideration to derive an architectural draft. The resulting hybrid architecture represents a balance between centralized and decentralized elements. The authors discuss the possible impact architectural decisions have on quality attributes such as transaction costs but without further validation. Rimba et al. [8] compare a business process in two execution environments: On the Amazon Simple Workflow Service cloud platform and on the Ethereum blockchain deployed as smart contract. Their results show costs two orders of magnitude higher for the blockchain variant than for the cloud platform. One drawback is that the architectures used in the example is either fully centralized or decentralized. There is no consideration of a hybrid architecture as a possible solution to lower costs.

B. Implementation

Design patterns are best practices for the implementation level represented as reusable code structures to solve reoccurring problems (cf. the ”Gang of Four” or ”GoF” patterns [9]). Using design patterns is a convenient way to increase code readability, lower redundancy and improve maintainability. In the context of blockchain-oriented software engineering [5] different motivations arise to use design patterns in SCs. Luu et al. show that a lot of different SCs suffer from the same types of issues and present a solution that relies on specific design patterns and best practices regarding security when creating SCs [10]. Zhang et al. illustrate the use of the GoF design patterns *Abstract Factory*, *Flyweight* and *Proxy* for implementing a health use case [11]. Furthermore, the authors highlight how the *Publish-Subscribe* pattern can support the application on an architectural level. Eberhardt and Tai [12] discuss off-chaining patterns as a means to reduce operational costs while maintaining the trustlessness of the implementation. Garcia-Banuelos et al. [13] examine the operational costs of multiple implementation variants of a business process. The authors use a business model described in BPMN (Business Process Model and Notation) and have Solidity code for an Ethereum smart contract generated. From this code two optimized variants are created. In one variant a new smart contract is deployed for each instance of the business process and for the other variant all instances are handled within a single smart contract. For the resulting implementation variants quality attributes such as gas costs (i.e., transaction costs) on the implementation level and throughput on the network architectural level are compared. Different random traces from the business process model are simulated and therefore represent multiple usage scenarios (e.g. invoicing with many process instances or a supply chain with many events during a single process instance). Using a single contract for all process instances results for example in the least gas costs. Best practices expressed by design patterns help to avoid security pitfalls, prevent vulnerabilities in SCs (cf. [14]) and help to deal with certain properties of blockchains. However, design patterns can also have a negative impact on resource usage. There are several studies examining the impact of design patterns on energy consumption, which is especially relevant when developing mobile applications [15]. Energy consumption is hard to capture because it requires specialized tools, e.g., to measure battery current during the execution of an application. Blockchain transactions have the advantage that the executed functions are deterministic by design. This allows for a predictable calculation of the gas being used and a very easy comparison of different implementations.

III. Architectural Blockchain Tactics

The idea to introduce architectural blockchain tactics is inspired by the work of Bass et al. [16] and Bachmann et al. [17] as an established way to describe the relation between architectural decisions and quality attributes. Their goal is to model how quality attributes are impacted by architectural decisions in order to control their outcome with suitable responses. Architectural tactics usually describe a given stimulus, multiple tactics to handle this stimulus and a response that should be achieved.

Our idea of architectural blockchain tactics is shown in Figure 1. The stimulus is in this case the intention to reduce the centrality of a system component or to reduce the required trust for interacting with a system. The envisioned blockchain tactics are separated in two aspects. The first aspect is the architectural design level for which we incorporate the approach by Wessling et al. [6]. The authors describe a process for deriving an architectural draft considering the participants of a system (users and existing systems), their trust relations and interactions. Their overall goal is to derive a hybrid architectural design which deals with blockchain properties, balances centralized and decentralized elements and thus minimizes the operational costs. The second aspect is the implementation level which is part of this paper and will be discussed in Section IV. For this aspect we propose to determine usage scenarios (”concrete scenarios” according to Bass et al. [16]) that are used to simulate operational costs for different smart contract implementations.

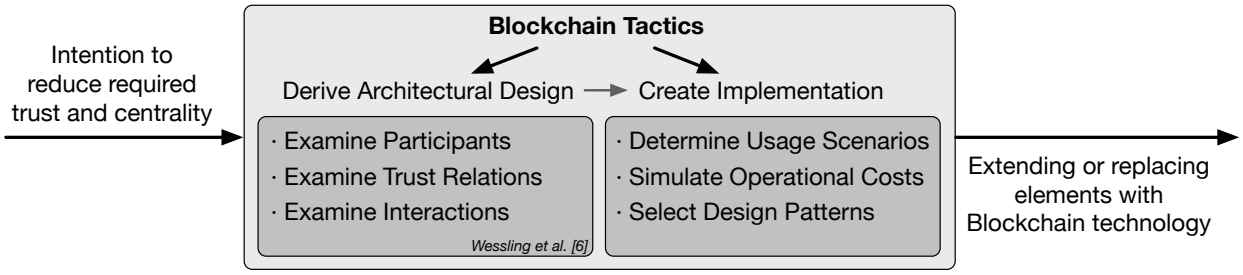


Fig. 1. Architectural Blockchain Tactics

The costs for using blockchain technology in an application appears to be a very important quality attribute (see. [8], [13]). In addition to the operational costs that we consider in this paper, there are several other facets of costs that can be identified, e.g., costs for engineering (development, deployment), maintenance (update, migration), etc.

In the next section we will focus on the implementation level and show that even the choice of design patterns can have a strong impact on transaction costs for an application. If there is already a noticeable impact on the implementation level, we argue that there must be an even bigger impact for architectural decisions on a broader level (e.g., defining modules or deciding which elements should be centralized or decentralized).

IV. Experiment

The goal of our experiment is to compare transaction costs of two smart contract implementations by simulating three different usage scenarios. We want to show the impact of implementation decisions (e.g., design patterns) on transaction costs. For the simulation we use the Ethereum blockchain where the total costs for a transaction consist of a fixed amount of gas per transaction and the variable execution costs, depending on the complexity of the called function. The simple SC used in this experiment allows to calculate the average of a set of numbers. The contracts are written in Solidity[1] and executed in the Remix Solidity IDE[2]. Two implementation variants were created to compare the costs with and without using a design pattern.

```
contract AvgOnDemand {
    int256 sum;
    int256 count;

    function add(int256 n) public {
        sum += n;
        count++;
    }

    function calc() public returns (int256) {
        return sum / count;
    }
}
```

Fig. 2. Smart contract with separate functions: Calculate average on demand

[1]Solidity v0.4.20, see https://solidity.readthedocs.io, visited on 2018-11-13
[2]https://remix.ethereum.org, visited on 2018-11-13

The variant without a design pattern is shown in Figure 2 and consists of two separate functions. To add a number to the overall sum, the function *add()* is used. Here the additional value *n* is saved and the counter is incremented. To gather the latest average value, the method *calc()* executes the calculation on demand and is called in a separate transaction.

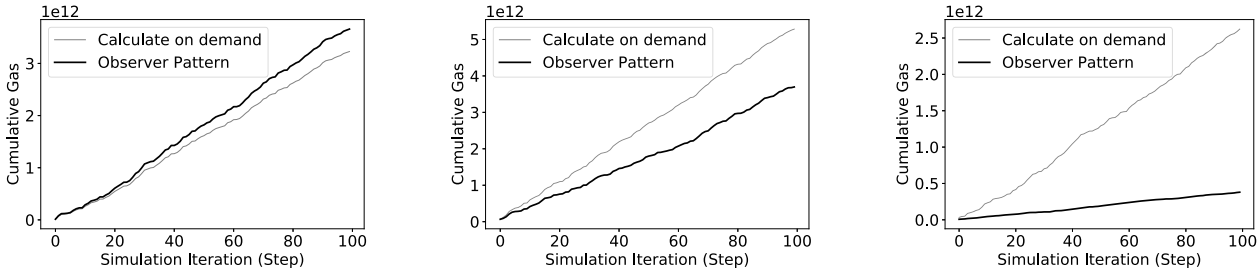
```
contract AvgObserver {
    int256 sum;
    int256 count;
    int256 avg;

    event UpdatedAvg(int256 value);

    function addCalc(int256 n) public {
        sum += n;
        count++;
        avg = sum / count;
        UpdatedAvg(avg);
    }
}
```

Fig. 3. Smart contract implementing the observer pattern using a combined function: Add number, calculate average and send event to notify observers

The variant in Figure 3 implements the observer pattern and combines both tasks in one function *addCalc()*. The parameter *n* is added to the sum, the counter is incremented and a new average value is calculated and written to the variable *avg*. Table I shows that the fixed gas amount per transaction is almost equal for all functions but that huge differences are visible for the execution costs. As a next step we have to consider the usage scenarios and compare the total transaction costs for both implementation variants. Table II shows the three fictional scenarios for this experiment. In Scenario A the task of adding a number is executed very often and calculating the average is executed less frequently. For scenario B both amounts are equal and for scenario C the amounts are the opposite of scenario A.

TABLE I
Transaction costs of smart contract functions for both variants

Variant	Function	Gas_{Fixed}	$Gas_{Execution}$
On Demand	*add()*	21464	10656
	calc()	21272	625
Observer	*addCalc()*	21464	17343



(a) Scenario A: Frequently adding numbers and infrequently calculating average

(b) Scenario B: Similar frequency adding numbers and calculating average

(c) Scenario C: Infrequently adding numbers and frequently calculating average

Fig. 4. Results of gas consumption for each scenario and both implementation variants

TABLE II
USAGE SCENARIOS AND AMOUNT OF FUNCTION CALLS

	Adding Number	Calculate Average
Scenario A	1k – 2000k	10 – 200k
Scenario B	10 – 2000k	
Scenario C	10 – 200k	1k – 2000k

The results of the experiment are shown in Figure 4. For each simulation step (x-axis) a new pair of random numbers is selected from both ranges given in Table II that represents the amount of repetitions for each task and is used for both implementation variants. The y-axis represents the cumulative transaction costs. The results of simulating scenario A (see Figure 4a) show that over time the calculation on demand results in less transaction costs than the observer pattern, as adding numbers is more frequent than calculating the average. For roughly equal amounts of adding numbers and calculating the average, the observer patterns turns out to be the better choice (see Figure 4b). Although the execution costs for the observer pattern is higher than for calculating on demand (see Table I), the latter requires two separate function calls and therefore two transactions with twice the fixed gas amount. Scenario C benefits from the observer pattern (see Figure 4c) as the average calculation is called very often and free of cost.

V. CONCLUSION AND FUTURE WORK

In this paper we explained the need for architectural blockchain tactics as a means to support the process of integrating decentralized elements in a software architecture and present a first version of its structure. We argue that decisions on both the architectural and implementation level have to be justified with appropriate metrics for quantifying quality attributes. We presented a simplified experiment to show the impact of design patterns in smart contracts on transaction costs. Research in the area of Blockchain-oriented Software Engineering (BOSE) is at an early stage [5]. Architectural blockchain tactics serve as a first step towards a coherent and fine-grained approach for employing distributed ledger technology and building hybrid architectures with the right balance between centralized and decentralized elements.

REFERENCES

[1] M. Swan, *Blockchain: Blueprint for a New Economy*. O'Reilly, 2015.
[2] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*, 1st ed. O'Reilly Media, Inc., 2014.
[3] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2013, visited on 2018-11-13. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper
[4] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," visited on 2018-11-13. [Online]. Available: http://gavwood.com/paper.pdf
[5] S. Porru, A. Pinna, M. Marchesi, and R. Tonelli, "Blockchain-oriented software engineering: Challenges and new directions," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 169–171.
[6] F. Wessling, C. Ehmke, M. Hesenius, and V. Gruhn, "How much blockchain do you need? towards a concept for building hybrid DApp architectures," in *IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 40th International Conference on Software Engineering ICSE 2018*. ACM, 2018.
[7] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba, "A taxonomy of blockchain-based systems for architecture design," in *IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 243–252.
[8] P. Rimba, A. B. Tran, I. Weber, M. Staples, A. Ponomarev, and X. Xu, "Comparing blockchain and cloud services for business process execution." IEEE, 2017, pp. 257–260.
[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., 1995.
[10] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 254–269.
[11] P. Zhang, J. White, D. C. Schmidt, and G. Lenz, "Design of blockchain-based apps using familiar software patterns to address interoperability challenges in healthcare," in *PLoP - 24th Conference On Pattern Languages Of Programs*, 2017.
[12] J. Eberhardt and S. Tai, "On or off the blockchain? insights on off-chaining computation and data," in *Service-Oriented and Cloud Computing*, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds. Cham: Springer International Publishing, 2017, pp. 3–15.
[13] L. Garca-Bauelos, A. Ponomarev, M. Dumas, and I. Weber, "Optimized execution of business processes on blockchain," in *Business Process Management*, ser. Lecture Notes in Computer Science. Springer, 2017.
[14] ConsenSys. Ethereum smart contract security best practices. [Online]. Available: https://consensys.github.io/smart-contract-best-practices/
[15] A. Noureddine and A. Rajan, "Optimising energy consumption of design patterns," in *Proceedings of the 37th International Conference on Software Engineering (ICSE) - Volume 2*. IEEE Press, pp. 623–626.
[16] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, ser. SEI series in software engineering. Addison-Wesley, 2003.
[17] F. Bachmann, L. Bass, and M. Klein, "Deriving architectural tactics: A step toward methodical architectural design," *Software Engineering Institute (SEI), Technical Report, No. CMU/SEI-2003-TR-004*, 2003.