Building Hybrid DApps using Blockchain Tactics - The Meta-Transaction Example

Florian Blum, Benedikt Severin, Michael Hettmer, Philipp Hückinghaus, Volker Gruhn
University of Duisburg-Essen
Schützenbahn 70, 45127, Essen, Germany
firstname.lastname@uni-due.de

***Abstract*—Building blockchain-based applications and deciding which elements of an architecture should employ blockchain technologies poses several challenges. Architectural design decisions have a strong impact on quality attributes such as privacy, operational cost, transparency, risk and user experience (UX). To deal with these challenges, we propose a structured approach using existing architectural concepts such as strategies, tactics and design patterns and illustrate their application using the Meta-Transaction design pattern. Meta-Transactions are cryptographically signed function calls (i.e. transactions) a user sends to a backend. The backend submits the transaction to the blockchain and pays the fees on behalf of the user. Due to the cryptographic signature, the backend is not able to manipulate the function name or its parameters, thus acting as a trustless proxy. Several other design patterns exist in the area of blockchain-oriented applications but it remains unclear how to decide which are suitable for a given use case and how quality attributes of the resulting system are affected. By using the Meta-Transaction design pattern as an example, this paper motivates why Blockchain Tactics and corresponding design patterns are necessary and help to structure best practices and common solutions for challenges of using blockchain technology.**

I. Introduction

Building applications using blockchain technology is a promising way for decentralizing elements of a software architecture in order to reduce the required level of trust and increase transparency of current systems [1], [2], [3]. Blockchain-Oriented Software Engineering (BOSE) is a growing discipline for creating such decentralized applications ("DApps") in a structured and effective way [4]. In addition to the aforementioned advantages, using blockchain technology also poses several challenges. The choice can have a strong impact on quality attributes of the resulting system, e.g. privacy, operational cost, transparency, risk and user experience. For many use cases it therefore makes sense to use blockchain technology only for certain parts when building an application, thus resulting in a "hybrid DApp" using blockchain and non-blockchain elements [5]. The decision which parts benefit from decentralization and which trade-offs have to be considered is not easy to make. There are certain types of information that need to be captured during requirements engineering in order to make that decision but are not considered in detail by current approaches (e.g. the level of trust or required level of privacy). A first step towards this goal are the approaches by Wessling et al. [5] or Marchesi et al. [6], who both present a software engineering process tailored to the specific needs of blockchain-oriented applications. Following a BOSE process, the software architect is constantly confronted with trade-offs due to challenges introduced by blockchain technology [4]. Those challenges can be categorized and refer to quality attributes such as user experience (e.g., requiring a wallet with cryptocurrency), privacy (e.g., storing personal information on a public ledger) and scalability (e.g., waiting for a transaction to be confirmed and added to the blockchain after each DApp interaction) [7], [8]. To handle those challenges, Blockchain Tactics and corresponding design patterns emerged to capture best practices and common solutions in a structured way [2].

In this paper we take a closer look at current Blockchain Tactics, describe the terminology around this area of research and use Meta-Transactions to give a first specific example of a design pattern implementing Blockchain Tactics. This paper is structured as follows. Section II gives an overview of current Blockchain Tactics and design patterns. In Section III we will introduce the concept of Meta-Transactions to explain the decision process when comparing different implementation options and their trade-offs. Section IV concludes this paper and gives an outlook on future work and open issues in this direction. For illustration purposes this paper refers to the Ethereum blockchain, as it is currently the most active environment [9], [10], although most of the concepts are independent from a specific blockchain implementation.

II. Blockchain Tactics

Architectural Tactics are introduced by Bass et al. [11] and Bachmann et al. [12] in the context of software architecture design and describe the relation between architectural decisions and quality attributes. Bass and Bachmann use three distinct terms to describe reoccurring challenges in the area of software architectures: Strategies, Tactics and Design Patterns. According to the authors, an Architectural Strategy describes an overall goal of the stakeholders (e.g., "reduce cost") and can be represented as a set of Architectural Tactics. Architectural Tactics follow the overall goal described by a strategy (e.g., "use cloud hosting" in order to reduce cost) and can also refer to more specific Tactics (as a refinement of a tactic) and design patterns (as an implementation of a tactic). A design pattern captures best practices and common solutions for reoccurring

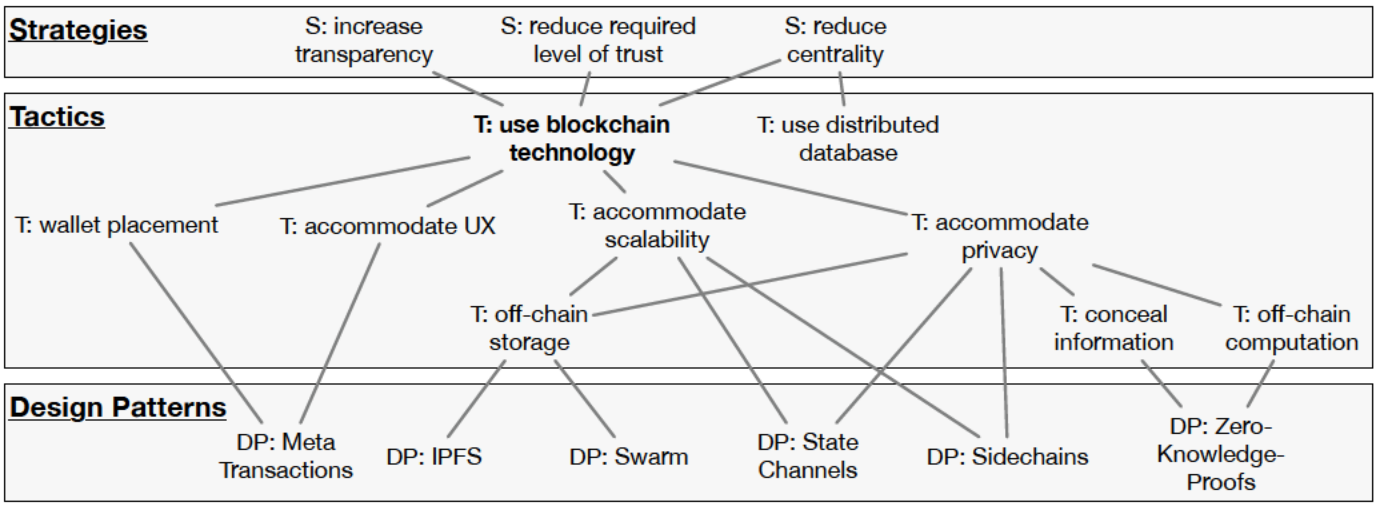

Fig. 1. Examples and relations of Strategies, Tactics and Design Patterns

challenges in a structured way (e.g. "use containers" as a more specific implementation of the tactic "use cloud hosting").

Blockchain Tactics by Wessling et al. adapt the idea of architectural tactics to systems using blockchain technology [2], as this decision introduces several challenges and can negatively influence quality attributes of the resulting system. The authors describe the general structure of a Blockchain Tactic which consists of a stimulus (e.g., the desire to reduce the centrality of a system), the response (e.g., a system with lower centrality) and a set of best practices to guide the execution of those desires on both architectural and implementation level.

This paper extends their work on Blockchain Tactics and examines the relation between Strategies, Tactics and Design Patterns in more detail. Figure 1 shows a few examples of Strategies, Tactics and Design Patterns in the context of blockchain-oriented applications that are currently emerging. Although this paper focuses on blockchain tactics, the structure of the figure is not restricted to blockchain technology as the strategy "reduce centrality" could also be achieved with the tactic "use distributed database". The figure also highlights the relation between those elements, e.g. what a possible stimulus is for each tactic, which sub-tactics exist, which design patterns support the implementation of a certain tactic, etc. The relations should highlight the implications between strategies, tactics and design patterns, as there is usually the need to balance overarching strategies with the result of certain tactic and design pattern choices. For example, in order to support the strategy "increase transparency" of the system, the "use blockchain technology" tactic is applied and therefore it is necessary to consider the impact on privacy. Here the stimulus of the sub-tactic "accommodate privacy" may refer to the challenge of storing personal data on a public blockchain. Sub-tactics like "off-chain storage" [13] with specific design patterns such as "IPFS"[1] and "Swarm"[2] exist and help to satisfy the privacy requirements while balancing the trade-offs. Literature regarding architectural tactics usually have an overlap between the terms strategy, tactic and design pattern. It is often hard to distinguish between tactics and design pattern, e.g. "microservices" can be seen as a tactic to decompose a business process but also a design pattern on how to implement fine-grained functional offerings. In general it can be stated that strategies are closer to the overall goals of the stakeholder and are therefore positioned on top of the figure. At the bottom of the figure, decisions are closer to the architectural level and often refer to technical components and details regarding the implementation. In our terminology, a strategy guides the engineering of a blockchain application while balancing quality attribute trade-offs. Tactics support the execution of strategies, are usually motivated by challenges imposed by using blockchain technology and focus on a single quality attribute. Design patterns help to implement tactics and should have a positive influence on quality attributes. In order to find the best combination of tactics for a given use case and stakeholder priorities, trade-off analysis methods such as ATAM [14] can be used.

[1]https://ipfs.io

[2]https://ethersphere.github.io/swarm-home/

III. Meta-Transaction Design Pattern

This section gives an overview of the Meta-Transaction design pattern. To illustrate the decisions and trade-offs, we use an example from a research project that uses blockchain technology to support subcontracting and payment handling for the building industry. In our use case, construction workers confirm a task on their mobile device, trigger a review process and then receive a partial payment for their current progress.

Blockchain technology can introduce a potential barrier for users to interact with the application. Wessling et al. [8] mention two opposing interaction types with different implications regarding the user's trust and necessary technological skill. On the one hand, users can create, sign and send transactions themselves, which requires no trust towards the backend but high technological skill. On the other hand, users trust the

backend completely and have it create, sign and send transactions on behalf of them, which requires no understanding of blockchain technology and results in high convenience and therefore an improved user experience. Interacting with a DApp usually requires a wallet with cryptocurrency in order to pay for transaction fees. This circumstance is the stimulus for the blockchain tactic "accommodate UX" which can be implemented based on the Meta-Transactions design pattern that is currently emerging. Meta-Transactions offer a trade-off between those opposing interaction types. It enables users to sign transactions on their device, having the backend submit it to the blockchain and pay for the transaction cost. The concept of Meta-Transactions is used in projects such as Status (IdentityGasRelay[3]), OpenZeppelin (ECDSA library[4]), uPort (proxy contract[5]), for the UniversalLogin[6] design pattern as well as the BouncerProxy[7] and MetaTx.io[8] by Austin Griffith. Those projects use Meta-Transaction to solve a common DApp problem: Submitting a blockchain transaction requires a transaction fee for which the users need cryptocurrencies. This is especially relevant for onboarding new users who are not familiar with blockchain technology. Currently, there are several concepts the user has to understand in order to interact with a DApp. Those concepts cover cryptography, the public-private keypair of a wallet, obtaining a cryptocurrency, understanding transaction fees ("gas costs" on the Ethereum blockchain), using tools such as the MetaMask browser plugin to interact with DApps, which browsers with built-in wallet functionality exist, etc. Meta-Transactions help lowering the onboarding barrier for users who are new to DApps by having the provider pay for transaction cost. This hides technical complexity but enables expert users to interact with the DApp independently as both forwarded Meta-Transactions and direct smart-contract interactions can be supported simultaneously.

The DApp provider has to decide for which interactions Meta-Transactions will be used. They can be used for all user interactions (which might increase cost due to additional smart-contract complexity) or to simplify the onboarding process. The design pattern also contributes to risk management for the operational phase. Paying transaction cost is also possible by sending cryptocurrency to the user's wallet upfront. However, leaving funds on multiple devices can increase risk and is hard to manage (cf. Figure 1: "wallet placement" tactic).

Figure 2 illustrates the three different options and which implications each has on UX, transparency and risk.

Option (A): Users are allowed to confirm tasks by calling a function on the backend (e.g. via REST API). The backend has a wallet with cryptocurrency funds, is able to send the confirmation to the blockchain and pays transaction fees. This option has an improved UX (no barrier for the user and details

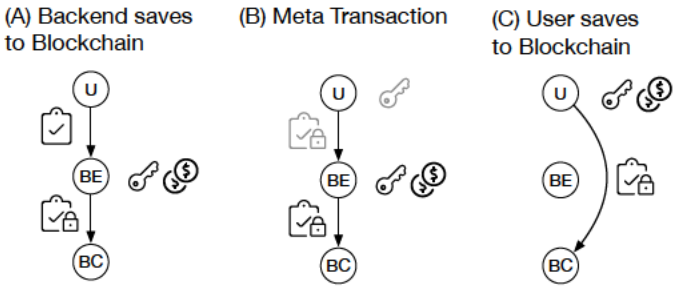

Fig. 2. Different options for implementing the use case

of blockchain technology are hidden), low transparency (users cannot be sure the task ID will not be changed and the backend signs as the origin of this information) and low risk for the operator (only a single wallet under its control).

Option (B): With the Meta-Transaction design pattern users are not required to have a wallet with cryptocurrency but are still able to create task confirmations and sign them with their private key. The backend checks the validity of the Meta-Transaction and sends it to the blockchain by paying transaction fees with a local wallet. This option offers an improved UX (users do not need cryptocurrency funds in their wallet), high transparency (Meta-Transactions cannot be manipulated and refer to the user as the true origin) and low risk for the operator (having only a single wallet to manage).

Option (C): Users can also send transactions directly to the blockchain. This option has a bad UX (users have to pay transaction fees by themselves and are required to have cryptocurrency funds), offers high transparency (users are visible as the origin) and high risk (users have to manage and secure their own cryptocurrency funds that are out of reach for the operator, distributed on many devices and hard to manage).

Option A and Option B allows the backend to refuse sending a transaction to the blockchain. Although a signed Meta-Transaction cannot be manipulated, the transaction can still be omitted in Option B. If that is an advantage or disadvantage depends on the point of view. For the DApp operator it is useful to prevent the submission of a transaction coming from a compromised device (e.g. in case it was stolen or is defect). This helps to prevent writing wrong information to the blockchain and reduces risk. For the user it might be a disadvantage as the backend operator is able to prevent a transaction from being sent to the blockchain. Nonetheless, this evaluation depends highly on the use case.

Figure 3 shows the interaction between user, backend and blockchain using Meta-Transactions. In our example, a user wants to confirm a task by calling the function `confirmTask(Task-ID)` on the smart contract of the DApp. Instead of sending a transaction directly to the blockchain, users sign the desired function call with their private key. This can be done in a web or mobile application by using the MetaMask[9] browser plugin, using a browser

[3]https://github.com/status-im/contracts/

[4]https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/cryptography/ECDSA.sol

[5]https://github.com/uport-project/uport-identity/

[6]https://github.com/UniversalLogin/UniversalLoginSDK

[7]https://github.com/austintgriffith/bouncer-proxy

[8]https://metatx.io

[9]https://metamask.io

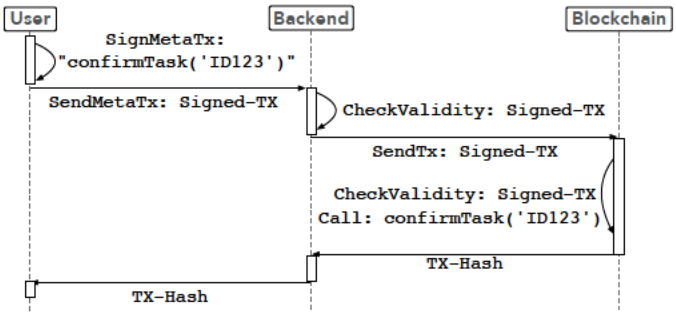

Fig. 3. Interaction sequence using Meta-Transactions

with built-in cryptocurrency wallet[10] or using a temporary wallet created in the background[11]. The user sends a signed transaction to the backend where its validity is checked and then sent to the blockchain by paying the transaction fee. Before sending it to the blockchain a first validity check can be done in order to prevent paying fees for invalid transactions. The function's name and input values are signed and therefore cannot be changed by the backend. The signature will also be checked on-chain within the smart contract before calling `confirmTask('ID123')`. Afterwards the backend receives the hash from the successful transaction and can return it to the user as a confirmation receipt. For a user there are two outcomes that both can be checked independently from and without trusting the backend provider: Either their transaction was added to the blockchain as intended (confirmed with the transaction hash) or not submitted at all. It is not possible to submit a manipulated transaction on behalf of the user.

```
1  function forward(bytes sig, address signer, address
       destination, uint value, bytes data) public {
2      // (1) check signature
3      require(signer == getSigner(_metaTxHash, sig));
4      // (2) execute function call
5      assembly {
6          call(gas, destination, value, add(data, 0x20),
               mload(data), 0, 0)
7      }
8  }
9
10 function getSigner(bytes32 _hash, bytes _signature)
       internal view returns (address){
11     // ...
12     return ecrecover(keccak256(
13         abi.encodePacked("\x19Ethereum Signed Message:\
               n32", _hash)
14     ), v, r, s);
15 }
```

Fig. 4. Solidity functions to check and forward a Meta-Transaction

The on-chain validity check of the Meta-Transaction is done with the Solidity function `ecrecover()`. Figure 4 shows the relevant parts of a smart contract that handles the validation. The function `forward()` in line 1 receives the signed transaction data from the user as parameter `sig` and is checked for validity in line 3. Here the function `getSigner()` from line 10 is called which uses the `ecrecover()` function to derive the origin (i.e., public address) of the given signature. The recovered origin has to match the user's address to confirm that the transaction is valid and has not been manipulated.

It needs to be considered that adding validity checks for Meta-Transactions within smart contracts results in higher complexity and increases transaction cost. To achieve the best trade-off between UX, transparency, risk and operational costs for the provider, the decision which function calls require this design pattern should be made carefully.

IV. Conclusion and Future Work

This paper discusses challenges of Blockchain-Oriented Software Engineering and motivates the need for Blockchain Tactics. Figure 1 gives a first impression of the relations between Strategies, Tactics and Design Patterns. The Meta-Transaction design pattern is used to explain that certain trade-offs regarding quality attributes of the resulting system have to be considered when comparing different implementation options for a specific scenario. In our scenario the use of Meta-Transactions served as the best trade-off between UX, transparency and risk but of course this conclusion is highly context-dependent. The frequency and complexity of function calls for a scenario have to be considered as using the Meta-Transaction design pattern has a strong impact on the resulting cost. We used a simple example and more complex scenarios and other design patterns need to be discussed in the future.

Current research regarding smart contract design patterns is at an early stage [15], [16]. Several best practices emerged as a general guideline when developing smart contracts [17], [18], but currently there are no guidelines for selecting those design patterns and examining which ones are suitable for a given use case. Approaches in the area of Blockchain-Oriented Software Engineering are currently emerging. Wessling et al. [5] and Marchesi et al. [6] both present a high-level process for building decentralized applications and explain which information are necessary to make well-founded architectural decisions. Nonetheless, both approaches are not specific enough to support design decisions on the architectural as well as the implementation level.

This paper serves as a first step towards such guidelines by identifying possible Strategies, Tactics and Design Patterns in the context of blockchain applications as a foundation for a coherent decision process. We motivated that using blockchain technology has a strong impact on quality attributes such as UX, transparency and risk and that further trade-off analysis (e.g. using ATAM [14]) is necessary to find the best combination of Tactics and Design Patterns for a use case.

[10]e.g. https://brave.com or https://www.opera.com/crypto

[11]e.g. implemented as https://github.com/austintgriffith/burner-wallet

References

[1] X. Xu, I. Weber, and M. Staples, *Architecture for Blockchain Applications*. Springer International Publishing. [Online]. Available: http://link.springer.com/10.1007/978-3-030-03035-3

[2] F. Wessling, C. Ehmke, O. Meyer, and V. Gruhn, "Towards blockchain tactics: Building hybrid decentralized software architectures," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C), Hamburg, Germany, March 25-29, 2019*, 2019.

[3] C. Ehmke, F. Blum, and V. Gruhn, "Properties of decentralized consensus technology – why not every blockchain is a blockchain," 2019. [Online]. Available: http://arxiv.org/abs/1907.09289

[4] 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.

[5] F. Wessling, C. Ehmke, M. Hesenius, and V. Gruhn, "How much blockchain do you need? towards a concept for building hybrid dapp architectures," in *1st IEEE/ACM International Workshop on Emerging Trends in Software Engineering for Blockchain, WETSEB@ICSE 2018, Gothenburg, Sweden, May 27 - June 3, 2018*, 2018, pp. 44–47.

[6] M. Marchesi, L. Marchesi, and R. Tonelli, "An agile software engineering method to design blockchain applications," 2018. [Online]. Available: http://arxiv.org/abs/1809.09596

[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 *2017 IEEE International Conference on Software Architecture*.

[8] F. Wessling and V. Gruhn, "Engineering software architectures of blockchain-oriented applications," in *2018 IEEE International Conference on Software Architecture Companion, ICSA Companion 2018, Seattle, WA, USA, April 30 - May 4, 2018*, 2018, pp. 45–46.

[9] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," 2013. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[10] A. M. Antonopoulos and G. Wood, *Mastering Ethereum: Building Smart Contracts and DApps*, 1st ed. O'Reilly Media, 2018.

[11] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, ser. SEI series in software engineering. Addison-Wesley, 2003.

[12] 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.

[13] J. Eberhardt and S. Tai, "On or off the blockchain? insights on off-chaining computation and data," in *Service-Oriented and Cloud Computing*, ser. Lecture Notes in Computer Science. Springer, Cham, 2017.

[14] R. Kazman, M. Klein, and P. Clements, "ATAM: Method for architecture evaluation," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-2000-TR-004, 2000.

[15] Y. Liu, Q. Lu, X. Xu, L. Zhu, and H. Yao, "Applying design patterns in smart contracts," in *Blockchain ICBC 2018*, ser. Lecture Notes in Computer Science, S. Chen, H. Wang, and L.-J. Zhang, Eds. Springer International Publishing, 2018, pp. 92–106.

[16] M. Wohrer and U. Zdun, "Smart contracts: security patterns in the ethereum ecosystem and solidity," in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, 2018, pp. 2–8.

[17] Ethereum. Solidity docs - common patterns. [Online]. Available: https://solidity.readthedocs.io/en/develop/common-patterns.html

[18] ConsenSys. Ethereum smart contract security best practices. [Online]. Available: https://consensys.github.io/smart-contract-best-practices/