Demonstrating the importance of well-defined design patterns in smart contract development

Michaël Verdonck
Faculty of Economics and Business Administration, Ghent University;
michael.verdonck@ugent.be;

Abstract. This paper demonstrates how the characteristics of blockchain lead to different design concepts and approaches in developing smart contracts compared to common design practices. Since blockchain technologies and smart contracts are a rather new development in the domain of information systems, little research has yet been performed in how such blockchain-based systems should be designed and integrated with existing technologies. Smart contract development and deployment will be a key feature in the creation of such blockchain-based systems. Therefore, it is the goal of this paper to briefly demonstrate how two unique characteristics of the blockchain – i.e. immutability and visibility – strongly affect the way a smart contract has to be developed compared to common development approaches. By increasing our understanding of these unique characteristics and how smart contracts are used and how they are implemented, we can facilitate future research efforts to design for instance new domain-specific languages that can aid developers with the specific challenges and vulnerabilities of smart contract development.

1 Introduction

While originally introduced as a technology to support new forms of digital currency [1], blockchain has evolved as a promising foundation to support any type of transactions in society [2, 3]. A blockchain can be seen both as a distributed database recording transactions between parties, as well as a computational platform to execute small programs – referred to as smart contracts [4]. More specifically, smart contracts can carry and conditionally transfer digital assets or tokens between parties. Since smart contracts are permanently stored on the blockchain, they can be publicly viewed and read by anyone, and can therefore be executed in a predictable and transparent way. These particular features of the blockchain technology enable certain advantages such as traceability, transparency and enhanced security [5]. Even though blockchain has been stated (or hyped) in the past couple of years as a stand-alone system that will provide a plethora of solutions to different problems, the blockchain technology is more likely to evolve in cooperation and co-existence with current information technologies and systems. Smart contract development and deployment will be a key feature in the creation of such blockchain-based systems.

However, due to the blockchain's singular features, developing smart contracts entails a rather different approach and design paradigm compared to developing scripts on more traditional and centralized information systems [6]. Since blockchain technologies and smart contracts are a rather new development in the domain of information systems, little research has yet been performed in how such blockchain-based systems should be designed and integrated with existing technologies. Therefore, it is the goal of this paper to briefly demonstrate how two unique characteristics of the blockchain – i.e. immutability and visibility – strongly alter the way a smart contract has to be developed compared to common development approaches. By exploring the unique characteristics of blockchain technologies, we can increase our understanding of this new technology, and determine the different design approaches that will result in efficient and secure blockchain-based systems.

2 Characteristics of Smart Contract Development

To demonstrate the impact of certain blockchain features on the development of smart contract scripts, this paper will focus on the Ethereum blockchain ecosystem. The reason for this particular blockchain is that Ethereum is the most prominent blockchain for smart contract development with a rich, Turing-complete development language called Solidity. Moreover, Ethereum is a public blockchain, as such possessing the singular and challenging features such as immutability and complete public visibility, in contrast to some private blockchains [7].

2.1 Immutability

A fundamental characteristic of blockchains – and hence smart contracts that are being deployed on the blockchain – is that they are immutable, meaning that they can never be modified or updated again. The initial code you deploy to a contract is there to stay, permanently, on the blockchain. This ensures some of the key advantages that blockchain has to offer, e.g. availability, a trusted decentralized ledger and traceability. However, this also results in several challenges when creating smart contracts, such as blockchain memory management or control. In order to reduce and optimize storage as much as possible, the Ethereum blockchain charges users to pay every time a function is executed, using a currency called gas. Users buy gas with Ether (the currency on Ethereum) and have to spend ETH in order to execute functions on a smart contract. The amount of gas required to execute a function depends on the complexity of the respective function. Each individual operation has a gas cost based roughly on how much computing resources will be required to perform that operation – e.g. writing to storage is much more expensive than adding two integers. The total gas cost of your function is the sum of the gas costs of all its individual operations. Since the execution of functions in Solidity costs real money for its users, code optimization is much more important in Solidity than in other programming languages. If the program code of a smart contract is carelessly written, users are going to have to pay a premium to execute its functions – which could add up to large, unnecessary fees across thousands of users.

As a result, developing a smart contract in Solidity requires careful consideration of the kinds of types one will use for their variables. For instance, when defining more complex data types such as *Structs* (which are the equivalent of objects in object-oriented programming languages such as Java), the choice of its properties can make a considerable difference. In the programming examples below, the Struct Person in example A is defined by a regular data type *uint* (equivalent of an Integer), which is stored to the Ethereum blockchain as 256 bits. Example B on the other hand explicitly defines a sub-type of uint that only requires 8 bits of storage and will save a substantial amount of gas when the number of users gain extensively.

```
// Example A
struct Person {
uint age;
string name;
}
```

```
// Example B
struct Person {
uint8 age;
string name;
}
```

Moreover, the Ethereum blockchain differentiates between functions that for example write to the blockchain, or that only perform read tasks on the blockchain. Consequently, functions in a smart contract can be declared as a read-only function by adding a *view* keyword to the function. This particular emphasis of the Ethereum blockchain to optimize storage as much as possible concerning functions and data types can also lead to counter-intuitive programming patterns. For instance, in most programming languages, repeatedly looping over large data sets is computationally inefficient. However, in Solidity this approach is much more desirable than using storage on the blockchain. For smart contract development, it makes more sense to call view functions that iterate every time again over the blockchain then for example simply saving the data in an array as a variable for quick lookups – which would be the more logical approach in standard program logic.

2.2 Public Visibility

A second major challenge concerning the development of blockchain applications and smart contracts is that all data that is stored on the blockchain is publicly visible by anyone. Since smart contracts are deployed on the blockchain, this means that the entire program code of a contract can be viewed by anybody. While this holds certain advantages – e.g. participants of a smart contract can always verify the code themselves and therefore check what kind of contract they are interacting with – this also results in certain challenges. One major challenge is related to the security of a smart contract. In the case of a crucial flaw in your contract code, there is no way to adopt or modify the contract later. The only solution would be to tell your users to start using a different smart contract that has the corrected code to the flaw. Evidently, this causes confusion and hinderance for users and should only be considered as a last-resort solution. In order to cope with these issues, smart contract developers need to heavily test their program code before deploying it to the blockchain.

Another consequence of the public visibility of blockchains is that functions – those of which are declared public – can now be called by anyone. However, certain functions require restricted access, since they can handle for instance sensitive information, monetary value or perform updates on the contract. To handle such specific cases, a unique practice in Solidity emerged where functions can be made Ownable – meaning that certain functions have an owner who has special privileges. The program code below demonstrates how a contract can implement ownership. Briefly explained, the person who deploys the smart contract on the blockchain is the first (and only) one to call the constructor of the smart contract. As a result, this person is assigned as 'owner' of the smart contract and can execute certain that are for instance only assigned to the owner. The function isOwner() can for example be deployed to verify that a certain user who calls a protected function, is in fact the owner of the contract.

```
// Enabling ownership of functions in Smart Contracts
constructor() internal {
  _owner = msg.sender;
}

function isOwner() public view returns(bool) {
  return msg.sender == _owner;
}
```

3 Conclusion

In this paper, we demonstrated how the characteristics of blockchain lead to different design concepts and approaches in developing smart contracts compared to common design practices. For instance, due to immutability, it makes more sense to compose a new search query on the entire blockchain than to store instances or objects in an array. Moreover, the public visibility of blockchains have led to new concepts such as ownership in order to safeguard certain functions in a smart contract from being called by anyone. While this paper only briefly discussed the characteristics of immutability and visibility, their impact and influence on the design of smart contracts can be examined much more extensively. Additionally, this paper did not review other characteristics of the blockchain and smart contracts such as external dependencies, payable functionalities and multi-signature operations. By increasing our understanding of these unique characteristics and how smart contracts are used and how they are implemented can facilitate future research efforts to design for instance new domain-specific languages that aid developers with the challenges and vulnerabilities of smart contract development as discussed above. An example of such a research effort is the one by Gal & McCarthy[1], where they defined a single agreed structure for the development of smart contracts based upon the REA ontology [8]. As more blockchain-based systems and applications will be developed in the coming years, the importance of such clearly defined development environments, design patterns and modeling languages will only increase.

[1] http://ceur-ws.org/Vol-2239/article_11.pdf

References

1. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. 1–9 (2008).
2. Azaria, A., Ekblaw, A., Vieira, T., Lippman, A.: MedRec: Using blockchain for medical data access and permission management. Proc. - 2016 2nd Int. Conf. Open Big Data, OBD 2016. 25–30 (2016).
3. Karamitsos, I., Papadaki, M., Barghuthi, N.B. Al: Design of the Blockchain Smart Contract: A Use Case for Real Estate. J. Inf. Secur. 09, 177–190 (2018).
4. Staples, M., Chen, S., Falamaki, S., Ponomarev, A., Rimba, P., Tran, A.B., Weber, I., Xu, X., Zhu, L.: Risks and opportunities for systems using blockchain and smart contracts. Data61 (CSIRO), May. (2017).
5. Bartoletti, M., Pompianu, L.: An Empirical analysis of smart contracts: Platforms, applications, and design patterns. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 10323 LNCS, 494–509 (2017).
6. Wöhrer, M., Zdun, U.: Design Patterns for Smart Contracts in the Ethereum Ecosystem. (2017).
7. Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C., Rimba, P.: A Taxonomy of Blockchain-Based Systems for Architecture Design. In: 2017 IEEE International Conference on Software Architecture (ICSA). pp. 243–252. IEEE (2017).
8. McCarthy, W.E.: The REA Accounting Model - A Generalized Framework for Accounting Systems in a Shared Data Environment. Account. Rev. 57, 554–578 (1982).