Preliminary Steps Towards Modeling Blockchain Oriented Software

Henrique Rocha
Inria Lille - Nord Europe
Villeneuve D'ascq, France
henrique.rocha@gmail.com

Stéphane Ducasse
Inria Lille - Nord Europe
Villeneuve D'ascq, France
stephane.ducasse@inria.fr

ABSTRACT

Even though blockchain is mostly popular for its cryptocurrency, smart contracts have become a very prominent blockchain application. Smart contracts are like classes that can be called by client applications outside the blockchain. Therefore it is possible to develop blockchain-oriented software (BOS) that implements part of the business logic in the blockchain by using smart contracts. Currently, there is no design standard to model BOS. Since modeling is an important part of designing a software, developers may struggle to plan their BOS. In this paper, we show three complementary modeling approaches based on well-known software engineering models and apply them to a BOS example. Our goal is to start the discussion on specialized blockchain modeling notations.

CCS CONCEPTS

• **Software and its engineering** → *Designing software*; Entity relationship modeling; Unified Modeling Language (UML); • **Applied computing** → Business process modeling;

KEYWORDS

Blockchain, Modeling, Smart Contracts, UML, BPMN, ER Model

ACM Reference Format:
Henrique Rocha and Stéphane Ducasse. 2018. Preliminary Steps Towards Modeling Blockchain Oriented Software. In *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3194113.3194123

1 INTRODUCTION

Blockchain has become very popular recently due to the general adoption of cryptocurrencies [12]. Blockchain provides a platform for monetary interactions without the need of a central trusted authority [1, 10, 12, 17]. Cryptocurrencies like Bitcoin [14] and Ether [6] are common enough to be recognized among most people.

Roughly speaking, "blockchain is a globally shared, transactional database" [7]. Although far from a traditional database, this simple definition can provide a good point-of-view on blockchain. The blockchain database is managed by a peer-to-peer network where all peers store a complete copy of the database. Each record in this database is a block that is linked to the previous one forming a sequence. The blocks are immutable, which promote trust because the records cannot be altered or deleted.

Even though blockchain technology is mostly recognized for its cryptocurrencies, it has been used for other applications as well [3, 4, 9, 11, 13]. A very prominent application of blockchain is managing smart contracts [12].

Smart contracts are programs written in Turing-complete languages that run on the blockchain platform [1, 10, 12]. If we follow the analogy that blockchains are like databases, then smart contracts are like stored procedures since they execute procedural programming in the blockchain data. However, a better analogy is to see smart contracts as classes, because they are composed of data attributes and functions [7]. Moreover, a contract can extend another through inheritance just like classes in object-oriented programming.

Once a contract is deployed in the blockchain, it can interact with other contracts or client applications. Client applications employ proxies to remotely call contract functions just as calling any other object. This simple way to interact with contracts allows developers to implement applications that combine standard developing with blockchains. For instance, we could implement all business logic in smart contracts while the user interface is implemented as a desktop or web application. Likewise, we could also create a hybrid blockchain application, where only a part of business logic uses blockchain. The term blockchain-oriented software (BOS) defines these types of applications that work with blockchain [19].

However, currently, there is no standard notation available to design or model BOS. A system using blockchain could need a specialized notation to represent it [19]. The lack of specialized notation can over-complicate the adoption or migration to BOS, since the interaction between the blockchain and the application will not be properly specified. In this paper, we present three complementary modeling approaches for BOS based on the following modeling standards: Entity Relationship Model, Unified Modeling Language, and Business Process Model and Notation. We also use a simple application scenario to illustrate our modeling as well as describing the advantages and disadvantages of each modeling approach. Our goal is to begin the discussion on the lack of specific modeling notations to specify BOS and provide a starting point for discussion and better notations.

The remainder of this paper is organized as follows. In Section 2, we present some basic concepts on the modeling standards that we use in this paper. Section 3 describes our BOS application example, its blockchain contract, and the three modeling approaches applied

WETSEB'18, May 27, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5726-5/18/05…$15.00
https://doi.org/10.1145/3194113.3194123

to specify parts of it. In Section 4, we compare and discuss the modeling for BOS. Section 5 presents the related work. Finally, Section 6 presents the conclusions and future work ideas.

2 BACKGROUND

In this section, we present basic concepts to three modeling standards that we employ on this paper: Entity Relationship Model (Section 2.1), Unified Modeling Language (Section 2.2), and the Business Process Model and Notation (Section 2.3).

2.1 Entity Relationship Model

The entity-relationship model (ER model) was originally proposed by Chen [2] and it is a popular high-level design for modeling data in relational databases [5, 20]. The ER model is used in the conceptual design for databases and it defines entities and the relationship among them. The ER model can also be extended to be used in a database's logical design. In such case, the model is enhanced with additional detail on how the data and the relations will be stored in the database. Moreover, we can graphically express the database logical structure and design defined in the modeling by drawing as a diagram [20].

2.2 Unified Modeling Language

Unified Modeling Language (UML) [16] is a general purpose standard to specify, describe, design, visualize, and document software systems. The UML standard is a unification of many object-oriented modeling notations from the 1980s and 1990s. Therefore, UML was specially designed to model software programmed using the object-oriented paradigm [8]. The Object Management Group (OMG) controls the UML standard since 1997.

UML defines 13 diagrams that can be classified into three categories: Structure, Behavior, and Interaction. UML structure diagrams are composed of Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram. Behavior diagrams include Use Case Diagram, Activity Diagram, and State Machine Diagram. Finally, the interaction diagrams are the following: Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram [8, 16].

2.3 Business Process Model and Notation

Business Process Model and Notation (BPMN) [15] is a graphical notation to specify business process. BPMN is also controlled by the OMG (i.e., the same group that controls UML). Roughly speaking, a BPMN presents a flow-oriented representation of a software requirement [18]. Therefore, BPMN is useful to describe and capture the functional behavior of a requirement.

3 MODELING

In this section, we present our modeling approaches applied to a BOS. First, we describe a BOS example and its smart contract as a scenario for our modeling (Section 3.1). Second, we present a data driven approach to model our example (Section 3.2). Third, we show a structure driven approach using UML class diagram (Section 3.3). Finally, we present a process driven modeling using BPMN (Section 3.4).

3.1 Application Example

We will use a simple BOS as an example for our modeling approaches. Let's suppose a store that wants to create a fidelity point program. The store already has a web application to manage and sell its products. By using smart contracts, the fidelity points can be exchanged freely among clients without the need of the store's involvement. We need to use blockchain accounts to increase the security on the contracts and link those accounts to the store's regular client database.

The fidelity points requirements are similar to the cryptocurrency example for smart contracts [7]. Listing 1 shows the contract with the basic functions for the fidelity points.

Listing 1: Solidity Store Points Contract

```
1  pragma solidity ^0.4.18;
2
3  contract FidelityPoints {
4     mapping (address => uint) private points;
5     address private owner;
6
7     event Sent(address from, address to, uint
          amount);
8
9     function FidelityPoints() public {
10       owner = msg.sender;
11    }
12
13    function createPoints(address receiver, uint
          amount) public {
14       assert(msg.sender == owner);
15       points[receiver] += amount;
16    }
17
18    function send(address receiver, uint amount)
          public {
19       assert(points[msg.sender] >= amount);
20       points[msg.sender] -= amount;
21       points[receiver] += amount;
22       Sent(msg.sender, receiver, amount);
23    }
24
25    function consumePoints(address consumer, uint
          amount) public {
26       assert(msg.sender == owner);
27       assert(points[msg.sender] >= amount);
28       points[consumer] -= amount;
29    }
30
31    function balance() public view returns (uint) {
32       return points[msg.sender];
33    }
34 }
```

Listing 1 starts by defining the Solidity version (line 1) and the contract definition (line 2). In Solidity, *address* is a primitive type that refers to an ethereum account (i.e., a user or another contract). Mappings are like hash tables, which we used to store the fidelity points (unsigned integers or uint) and using the client addresses as key (line 4). Moreover, we created an event to allow applications to react to changes performed by this contract (line 7). The constructor (lines 9-11) stores the contract creator for security checks later. The rest of the contract defines the following functions:

- *createPoints* (lines 13-16). This is the only function that creates new points, therefore only the contract owner can invoke it. The assert function checks for a condition and throws an exception if such condition is not met. In this particular function, we assert that the person executing it is the contract owner.
- *send* (lines 18-23). This function allows one client to send his points to another one. We assert that the user does not spend more points than he/she has. We also call the Sent event, which we previously defined, to allow applications to react to the transfer.
- *consumePoints* (lines 25-29). This function allows the store to consume points from one client (probably because the client spent his fidelity points). We assert that only the contract owner can invoke this function and also that the client has sufficient points to be consumed.
- *balance* (lines 31-33). This function allows the client to check his balance in our fidelity points. Since this function does not change the state of the contract as it only returns a stored value, we marked the function with the view keyword so it does not cost anything for the client to execute it.

In a real software development scenario, it would be better to plan and design the application before coding. However, since there aren't specific modeling notations or tools for BOS, we started coding the contract based on our scenario requirements. We want to model not only the smart contract but also its interactions with our application.

3.2 Data Driven

Since blockchain is like a database, we could try to model it focusing on its data. Therefore, we can specify the data in a BOS using an ER model for the conceptual and logical design. If the BOS also uses a relational database in its application (a common scenario) then we can easily enhance the ER model for the relational database with the blockchain data.

This data driven modeling approach has the advantage to be easy to understand, use and capture data. Since ER modeling is a very popular standard, most software engineers are already familiar with its design. Another advantage is the possibility explicitly model the link between the blockchain and the relational database. The main disadvantage is that ER model can only capture data and it is not able to model the functional structure and behavior. An important part of a smart contract is not only data but also its functions and behavior.

For example, lets model our contract data using an ER model (Figure 1). Our smart contract needs to store the owner of the contract and a list containing the points for each client. We model the list as a one-to-many relationship in the ER model, but in the contract, we implemented that as a *mapping* (Listing 1, line 4). The mapping uses the client blockchain id (a primitive address type) as a key to access the points. We also create a relationship between the points in the blockchain and our private Client database, so that we can keep track on our main application.

When we extend the ER model conceptual design (Figure 1) for the logical design, we can specify the implementation of blockchain and relational database artifacts for each entity (Listing 2). For

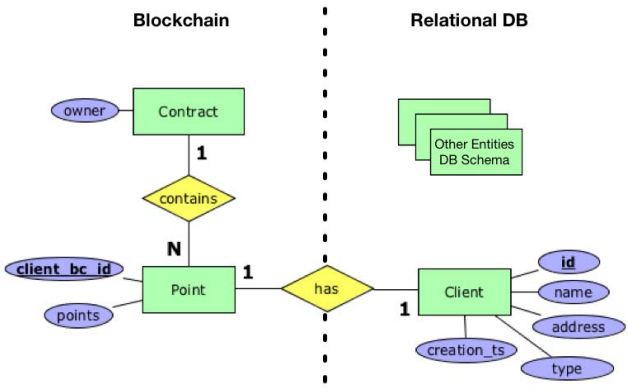

Figure 1: Store Points contract ER model example.

instance, we can specify the Point entity as a mapping in the logical design (line 3). We can also place a foreign key to implement the relationship between Point and Client (line 9, bcAddress attribute). Therefore, the ER model can be used to specify the data in BOS. On the other hand, we can see that most of the contract code (Listing 1) is functions and very little is used for data. Thus, an ER model is insufficient to design all aspects of a BOS.

Listing 2: Store Points ER model logical design

```
1 --- Blockchain
2 CONTRACT ( owner: address,
3     point: mapping address=>uint )
4
5 --- Relational DB
6 CLIENT (id int primary key, name varchar(100),
7     address varchar(100), type int,
8     creation_ts timestamp,
9     bcAddress byte(20) foreign key)
```

3.3 Structure Driven

The UML notation has six diagrams to model the structure of object oriented systems. Since smart contracts are very similar to classes, we can use UML diagrams with little adaptation. Therefore we can model an object oriented application and its blockchain structure by using UML. One advantage of using UML structure diagrams is that we can easily model and specify the functions and data attributes on smart contracts. Moreover, since UML is a popular modeling standard, it is easy for software engineers and developers to understand it. The disadvantage of structure modeling is that we can not specify the functional behavior of business process; for that we would need behavior models or a methodology focused on process.

For example, we present our contract (Listing 1) modeled as a UML class diagram (Figure 2). The class diagram represents the internal structure of the contract. In this example, we also model a Client entity class similar to the ER model we showed earlier (Figure 1). In fact, we placed the attribute *bcAddress* in the Client entity because of the relationship in the ER model (between Point and Client). As we can see, there is no relationship between the Client class and the contract, because the class diagram cannot capture the

link between the data as the ER model. Moreover, class diagrams relationships (e.g., aggregation, composition, etc.) may mislead developers on how to implement the interaction with the blockchain and code superfluous (and costly) objects into the blockchain.

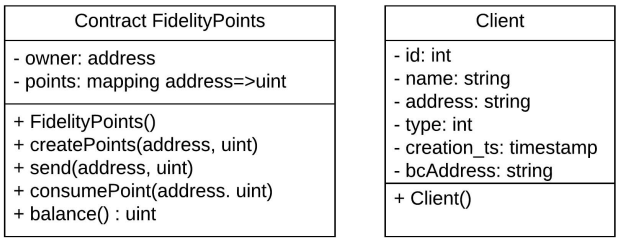

Figure 2: Store Points contract and client class diagram example.

In Figure 2, we did not model other classes to not clutter the diagram. However, if we model more classes and contracts together in the same diagram, then we would need a special notation to differentiate them for a better visualization. For instance, consider that we improved the diagram with more classes for the application domain (Figure 3). We used a small *"chain"* icon in the contract graphical representation as a notation to more easily identify it as a blockchain artifact.

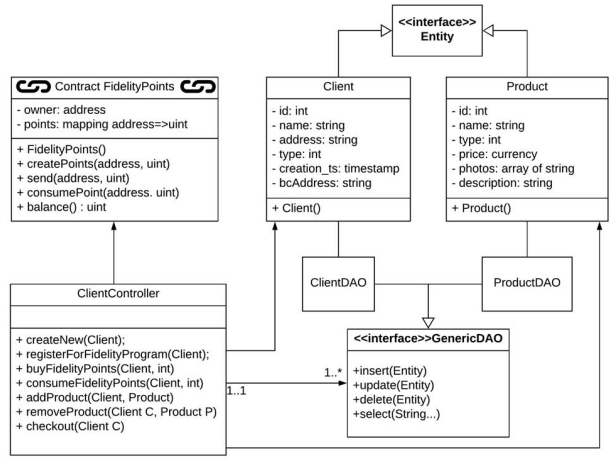

Figure 3: Enhanced contract and client class diagram example with special blockchain notation.

3.4 Process Driven

Although very useful, both Data and Structure Driven modeling approaches cannot specify details of a process. When we deal with BOS, a business process may need a detailed specification to help developers and engineers implement it. BPMN is an appropriate notation to specify business process. The main advantage is that we can specify the process behavior easily. On the other hand, we are not able to model complex data using BPMN. Another disadvantage is that it is difficult to model an overview of the software using BPMN, as we are focusing on specifying individual business process.

For example, let's consider the business process of a client already registered in our online store web application, and such client wants to register for our blockchain fidelity program. We can use BPMN to specify the process (Figure 4) and the *swimlane* notation (i.e., the named box container also referred as *pool*) to specify interactions with blockchain. We could also use an icon (similar to the one we used for Figure 2) to highlight blockchain tasks.

4 DISCUSSION

As we can see from the previous section, all models have their advantages and disadvantages when specifying BOS. The ER model (Figure 1) is good at capturing the data elements and their relationship. In fact, data driven modeling like ER model might be the only suitable notation to specify data relationship between blockchain and a private database. Since the data on blockchain is publicly accessible, it is important to not expose important or sensible data on smart contracts. Therefore, we carefully decide what to place on the blockchain and then link the contract data to our private (and possibly more secure) database. In our ER model example (Figure 1), the clients' blockchain address and their amount of points are exposed. However, it is not possible to acquire the clients' name and home address without accessing our private database. Both blockchain and private database data are linked by the relationship between the Point and Client entities, which translates to a foreign key into the Client table (Listing 2, line 9).

The class diagram (Figure 2) can model the internal structure of smart contracts. When we compare to the ER model, it has the advantage to model not only data attributes but also the functions. Since smart contracts functional behavior could be as important as its data, we might need to model all the internal structure of a contract. On the other hand, a class diagram is not suitable to model all data relationship in a BOS. The relationship between the client blockchain address (stored in the contract) and the Client entity on the private database (Figure 1, and Listing 2) could be properly modeled in the class diagram.

The BPMN notation (Figure 4) is best suited to specify business process. Since the process behavior itself, in a BOS, could require a more detailed design specification. When we compare the BPMN to a class diagram or even an ER model, we can see that BPMN cannot properly model complex data or the internal structure of BOS artifacts. However, the *swimlane* notation is useful to model the interaction between the application and the blockchain. Indeed, BPMN might be the notation that requires least adaptations to properly model a BOS business process.

5 RELATED WORK

Porru et al. [19] argue that we need to develop specialized techniques for BOS. The authors describe issues and challenges faced when working with blockchain from a software engineering point-of-view, and they propose ideas and practices to improve the state-of-art on BOS. One of the ideas suggested by the authors was the need of specialized modeling for BOS. More specifically, they argued that existing models could be adapted to better specify blockchain.

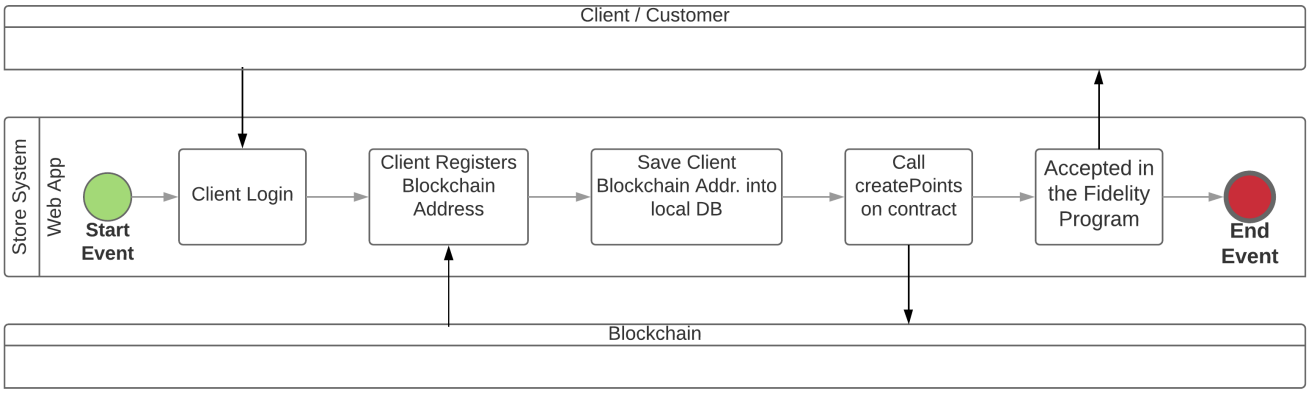

Figure 4: Client registers for the fidelity program BPMN example.

This was the inspiration for our work were we modeled a simple BOS example, which showed that current models are insufficient to specify all aspects of BOS. We also suggest simple adaptations to improve the modeling, but our main goal it is to start the discussion on the modeling of BOS.

We could not find many research aimed at BOS modeling. Xu et al. [21] argue that blockchain technology has many configurations and variants and they present a taxonomy based on blockchain properties and flowchart. They propose to use such elements to guide and build an initial checklist. Such properties are for example authority, storage and decentralization. Their outcome is a configuration of a blockchain system with technical details such as block creation time, block size, consensus algorithms. We plan to introduce the concerns of authority, storage and trust relationships as important property to be represented by our modeling approach.

As far as we know, our paper is the one of the first to begin modeling blockchain software. Most research related to smart contracts and blockchains leans towards security. For example, Luu et al. [12] investigate the security problems of executing Solidity smart contracts and propose solutions to make the contracts more secure. Bhargavan et al. [1] proposed a framework to convert smart contract to their own functional language F*, which was design to better verify the correctness and security of smart contracts. Juels et al. [10] analyse criminal smart contracts that are unsecured contracts and practices and raising the awareness of developing countermeasures against criminal contracts.

6 CONCLUSION

The popularity of cryptocurrencies made blockchain a hot topic among common people, practitioners, developers, and researchers. Another prominent blockchain application is smart contracts; programs that store a state and execute functions. By using smart contracts, we can develop software that maintains part of its data or logic in the blockchain. For this type of software, we use the term blockchain-oriented software (BOS).

In this paper, we present a simple example of a BOS, an online store creating a fidelity point program based on blockchain. For the BOS example, we took three modeling routes each focusing on one particular aspect: data driven, structure driven, and process driven. For the data driven, we created an ER model which is mostly used to specify data in relational databases. For the structure driven, we selected the class diagram as our model among all six UML structure diagrams. For the process driven, we used the BPMN notation. Every approach has its strengths and weakness, and a specialized notation for BOS is needed to properly design it. Our goal in this paper is not to propose a general solution, but to start the discussion and raise awareness to the lack of modeling notations for BOS.

We are working on the following directions for future work: (i) use a real BOS software development, to model and document its design process; (ii) verify if the behavior and interaction UML diagrams may also need adaptations to properly specify BOS; (iii) create a tool support for modeling BOS, as well as reverse engineer code into models, and use models to auto-generate code.

ACKNOWLEDGMENT

This work was supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council, CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020. This research was also supported by UTOCAT.[1]

REFERENCES

[1] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. 2016. Formal Verification of Smart Contracts: Short Paper. In *2016 ACM Workshop on Programming Languages and Analysis for Security (PLAS '16)*. ACM, New York, NY, USA, 91–96. https://doi.org/10.1145/2993600.2993611

[2] Peter Pin-Shan Chen. 1976. The Entity-relationship Model&Mdash;Toward a Unified View of Data. *ACM Trans. Database Syst.* 1, 1 (March 1976), 9–36. https://doi.org/10.1145/320434.320440

[3] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 1085–1100. https://doi.org/10.1145/3035918.3064033

[4] Stefan Dziembowski. 2015. Introduction to Cryptocurrencies. In *22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 1700–1701. https://doi.org/10.1145/2810103.2812704

[1] https://www.utocat.com/en, verified 2018-03-06

[5] Ramez Elmasri and Shamkant Navathe. 2010. *Fundamentals of Database Systems* (6th ed.). Addison-Wesley Publishing Company, USA.

[6] Ethereum Foundation. 2014. Ethereum's white paper. (2014). https://en.wikibooks.org/wiki/LaTeX/Bibliography_Management

[7] Ethereum Foundation. 2017. Solidity Documentation Release 0.4.18. (2017). https://media.readthedocs.org/pdf/solidity/develop/solidity.pdf

[8] Martin Fowler. 2003. *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3 ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[9] Adiseshu Hari and T. V. Lakshman. 2016. The Internet Blockchain: A Distributed, Tamper-Resistant Transaction Framework for the Internet. In *15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. ACM, New York, NY, USA, 204–210. https://doi.org/10.1145/3005745.3005771

[10] Ari Juels, Ahmed Kosba, and Elaine Shi. 2016. The Ring of Gyges: Investigating the Future of Criminal Smart Contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 283–295. https://doi.org/10.1145/2976749.2978362

[11] Benjamin Leiding, Parisa Memarmoshrefi, and Dieter Hogrefe. 2016. Self-managed and Blockchain-based Vehicular Ad-hoc Networks. In *2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct (UbiComp '16)*. ACM, New York, NY, USA, 137–140. https://doi.org/10.1145/2968219.2971409

[12] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. Making Smart Contracts Smarter. In *CCS'2016 (ACM Conference on Computer and Communications Security)*.

[13] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. 2015. Demystifying Incentives in the Consensus Computer. In *22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15)*. ACM, New York, NY, USA, 706–719. https://doi.org/10.1145/2810103.2813659

[14] Satoshi Nakamoto. 2009. BitCoin: A peer-to-peer electronic cash system. (2009). bitcoin.org

[15] Object Management Group. 2011. Business Process Model And Notation. (2011). http://www.omg.org/spec/BPMN/2.0 Version 2.0, document formal/11-01-03.

[16] Object Management Group. 2015. Unified Modelling Language. (2015). http://www.omg.org/spec/UML/2.5 Version 2.5, document formal/15-03-01.

[17] Russell O'Connor. 2017. Simplicity: A New Language for Blockchains. In *Proceedings of the 2017 Workshop on Programming Languages and Analysis for Security (PLAS '17)*. ACM, New York, NY, USA, 107–120. https://doi.org/10.1145/3139337.3139340

[18] Chun Ouyang, Marlon Dumas, Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Jan Mendling. 2009. From Business Process Models to Process-oriented Software Systems. *ACM Trans. Softw. Eng. Methodol.* 19, 1, Article 2 (Aug. 2009), 37 pages. https://doi.org/10.1145/1555392.1555395

[19] Simone Porru, Andrea Pinna, Michele Marchesi, and Roberto Tonelli. 2017. Blockchain-oriented Software Engineering: Challenges and New Directions. In *Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C '17)*. IEEE Press, Piscataway, NJ, USA, 169–171. https://doi.org/10.1109/ICSE-C.2017.142

[20] Abraham Silberschatz, Henry Korth, and S. Sudarshan. 2011. *Database Systems Concepts* (6 ed.). McGraw-Hill, Inc., New York, NY, USA.

[21] X. Xu, I. Weber, M. Staples, L. Zhu, J. Bosch, L. Bass, C. Pautasso, and P. Rimba. 2017. A Taxonomy of Blockchain-Based Systems for Architecture Design. In *IEEE International Conference on Software Architecture (ICSA)*. 243–252. https://doi.org/10.1109/ICSA.2017.33