Data modelling for Blockchain Oriented Software Engineering

Patrik Rek, Muhamed Turkanović
Faculty of Electrical Engineering and Computer Science, University of Maribor
Koroška cesta 46, Maribor, Slovenia
{patrik.rek,muhamed.turkanovic}@um.si

Abstract. *Blockchain Oriented Software Engineering (BOSE), which includes the specifics of blockchain ledgers, smart contracts, tokens and decentralized applications, is becoming an interesting topic due to its ever growing popularity, both to the scientific, as well as the professional community. Although there are already some defined aspects and good practices in the design and modelling of blockchain oriented software, these aspects mainly cover the structure and process perspectives. However, since blockchain is a form of a distributed and decentralized database, the BOSE has an evident shortcoming of covering only the data perspective of BOSE. The paper focuses on this shortcoming by proposing an extension towards the Entity-Relationship model, which covers the core data aspects of modelling blockchain-oriented software. The proposed enhanced data model was validated on a real-world use case.*

Keywords. blockchain, software engineering, modelling, smart contracts, ethereum, ER model, entity relationship, token, ledger

1 Introduction

Blockchain, supported with smart contracts, is becoming more and more popular as a platform for decentralized applications for various sectors. Decentralized applications are applications for which business logic depends on a blockchain platform, mainly, but not exclusively, supported by smart contracts. Decentralized applications, also called dApps, are mainly web applications where the back-end is an integrating blockchain and/or smart contract features, while the front-end, in the majority of cases, enables a connection to dedicated digital wallets, which enable users to authenticate and trigger actions based on their decentralized identities (e.g., blockchain addresses). Due to the specific nature of design and development of blockchain-oriented applications i.e., dApps, classical software engineering is not entirely applicable, hence IT architects and developers have to confront it with specific approaches. This requirement brought about the need for a dedicated approach, called Blockchain Oriented Software Engineering (BOSE) (Porru, Pinna, Marchesi, & Tonelli, 2017).

BOSE presents a challenge for many engineers trying to incorporate existing approaches to the blockchain. The first issues are the existing modelling techniques, which are not customized for BOSE, and therefore do not offer enough options for engineers to design appropriate models. There is no standardized solution for modelling blockchain-oriented software.

In this paper, we have addressed this challenge from a data perspective, which, in comparison to the process and structure perspectives, is not yet addressed appropriately for blockchain-oriented software.

1.1 Methodology and contributions

A literature review was conducted on existing approaches for BOSE, and specifically its connection towards modelling techniques. The literature review is documented in sections 2 and 3.

As the core contribution, we propose a modelling technique for BOSE focused on the data perspective, which is presented in section 4.

A validation of the proposed data modelling technique is provided through a real-world case, presented in sections 4 and 5.

2 Related works

Our paper addresses modelling techniques for blockchain applications. Therefore, a research was conducted of existing work done on that field. Since blockchain modelling is not yet addressed in many researches, this paper recognises only two existing papers.

Marchesi, Marchesi and Tonelli have addressed the agile blockchain decentralized application engineering (Marchesi, Marchesi, & Tonelli, 2020). They proposed the usage of agile software development practices because they are suited for systems with unclear starting requirements and changing systems. They have based their proposed system ABCDE on Scrum, and separated activities to blockchain oriented, such as smart contracts, libraries and data structures, and out-of-chain components, such as web or mobile applications. Together these activities form the design of a decentralized application. They advised usage of UML diagrams to help with the designing and modelling of software engineering. They used a UML class diagram for structural purposes, and a UML sequence diagram for communication purposes. Both UML diagrams were upgraded by the authors with stereotypes specific to BOSE.

Meanwhile, Rocha and Ducasse have addressed the issue of modelling blockchain-oriented software and the deficit of a standardized solution. They have covered the data, structure and process perspectives of blockchain-oriented software modelling using traditional modelling techniques. They also used a refined UML class diagram for structure modelling with additional stereotypes. However, the authors have not refined an ER diagram for data modelling, but used a traditional ER model notation (Rocha & Ducasse, 2018).

Other articles related to this paper address software modelling for traditional software, which doesn't have specific focus on blockchain oriented software. This article is by Rumbaugh and Booch, who addressed the UML in detail, with recommendations for application in structural, behavioral and architectural modelling in software engineering (Rumbaugh, Jacobson, & Booch, 2004).

While Rumbaugh and Booch address structure software modelling, Gorman and Choobineh focus on data modelling. They apply a traditional Chen's ER diagram on object-oriented software with its specifics (Gorman & Choobineh, 1990).

3 Background

3.1 Blockchain

Blockchain was first introduced by Satoshi Nakamoto in 2008 as an online decentralized and distributed ledger providing transparent data sharing (Nakamoto, 2009). Each of the transaction data, which are generated and stored in blockchain network nodes, are compressed and added to blocks. Various data are stored in distributed blocks. All the nodes containing the approved and prepared blocks form a chain of recursively connected blocks, thus, a blockchain. Once data are inserted into the blockchain, they becomes immutable, allowing verification of data on each block. Each operation is open to the public, transparent, and secure (Cheng, Lee, Chi, & Chen, 2018).

Blockchain provides different areas of usage. The most significant use case of blockchain technology is in the financial sector with crypto-currencies. There are more than 2,000 crypto-currencies currently on the market. The first among them was Bitcoin, introduced by Satoshi Nakamoto as a peer-to-peer electronic cash system (Nakamoto, 2009).

Following Bitcoin, which was defined as Blockchain 1.0, Ethereum was introduced by Vitalik Buterin as Blockchain 2.0, with support for smart contracts, which is, nowadays, a widespread blockchain use case (Aleksieva, Valchanov, & Huliyan, 2020; Wood, n.d.).

Ethereum is a decentralized platform, including Turing completeness with different options for applications. Most smart contracts are created using Ethereum blockchain networks. Bitcoin is considered a global payment network, while Ethereum may be considered a global computing system, due to its Ethereum Virtual Machine (EVM), which is located on every Ethereum blockchain node, and acts as the virtual machine for running smart contracts (Cheng et al., 2018).

3.2 Smart contracts

Ethereum enables developers to create applications which are stored in Ethereum-based blocks and are running on the EVM. These applications are incorruptible, secure and permanent, and are called smart contracts (Cheng et al., 2018).

Smart contracts are executed with the transaction validation. Deployment of the smart contract is executed as a special transaction, which sends the smart contract to the ledger. When the transaction is executed, a smart contract is assigned a unique address, and its code is uploaded to the blockchain. Smart contracts consist of an address, balance, executable code and state. Users of the blockchain may interact with a contract by sending transactions to a known contract address. Users can read or update the state using these interactions, interact with other contracts, or transfer value to others. Transactions can include the execution fee. When the transaction is accepted, all network participants execute the contract code. Afterwards, they agree on the output and the next state of the contract (Wohrer & Zdun, 2018).

3.2.1 Ethereum Virtual Machine

A decentralized virtual machine, which handles computer tasks and the state of smart contracts is called EVM. It is a network of smaller discrete machines in constant communication. All transactions executing the smart contracts are handled locally on each node and processed synchronously. Each node validates and groups the transactions in blocks, and tries to append them to the blockchain to collect a reward. This is known as mining. Every operation on the EVM has a cost, measured in units of gas. Operations that are computationally more demanding cost more gas than easier operations. This ensures the system is not overloaded. A transaction fee in Ethers is paid as the gas (Wohrer & Zdun, 2018).

3.2.2 Solidity

EVM operates in bytecode, which is a low-level operation instruction. Most smart contracts are written in higher-level programming languages, which are later compiled to EVM bytecode. There are more such languages, but the most prominent and widely adopted programming language for smart contracts is Solidity (Wohrer & Zdun, 2018).

Solidity is a high-level Turing-complete programming language with a JavaScript similar syntax. It is typed statically, supports inheritance and polymorphism, as well as libraries and complex user-defined types. Smart contracts in Solidity are structured similarly to classes in object-oriented programming languages. The code consists of variables and functions, which manipulate them, like in traditional imperative programming (Wohrer & Zdun, 2018).

Functions in Solidity can be designated as private or public, which manages the external access. Variables can be of different types, which are similar to traditional imperative programming languages. Specific types in Solidity are (1) Address, (2) Members of address, like balance and transfer, (3) Struct, (4) Mapping, and (5) Events (Dannen, 2017).

Additionally, Solidity defines special variables (`msg`, `block`, `tx`), which are always accessible in the global namespace, and contain information about the transaction and the blockchain. A developer may obtain complete call data, remaining gas, the transaction's sender, the value of the transaction, gas price, current block number, current block timestamp and current block miner's address. (Dannen, 2017; Wohrer & Zdun, 2018).

Another Solidity difference to JavaScript is modifiers and events. Modifiers are enclosed code units that modify functions' code execution flow. They allow condition-oriented programming that removes conditional paths in function bodies. Modifiers are specified after the function name. They are used to check conditions before function execution. Events are dispatched signals that smart contracts can fire. Different applications can contain listeners to these events without cost. Events also help with logging, since they are all contained in a transaction's log and their arguments. The transaction's log is a special data structure in the blockchain that connects all the way up to the block level. (Wohrer & Zdun, 2018).

3.3 Software modelling

Software engineering is the process of design, development, testing and delivery of software to final customers. A large part of software engineering is software modelling. Models are built and analysed before implementation for achieving the most acceptable final result (Gomaa, 2011).

Since blockchain smart contracts may be different for various areas and applications, the agile software development approach such as SCRUM, or non-agile approach such as waterfall, may be used (Beck et al., 2013; Balaji & Murugaiyan, 2012).

Both the approaches utilize software modelling in a large amount. Graphic representation of models helps in communicating the different views on software. A better understanding of software can be obtained by considering different perspectives, which is best represented with graphical models (Gomaa, 2011).

There are three perspectives of software modelling, which we will address separately. (Rocha & Ducasse, 2018):

- structure,
- process and
- data.

3.3.1 Structure software modelling

The main deficiency in data software modelling for blockchain-oriented software are functions, which can be represented using structure software modelling. The main Standard to specify, design, visualize and document software system from a structural perspective is Unified modelling Language (UML) (Rocha & Ducasse, 2018).

It unifies more object-oriented modelling notations. The Object Management Group (OMG) controls the UML Standard, which defines 13 diagrams classified into three categories (structure, behavior and interaction). UML structure diagrams consist of a class diagram, an object diagram, a component diagram, a composite structure diagram, a package diagram and a deployment diagram. For modelling smart contracts, UML class diagrams may be used with a few modifications (Rumbaugh et al., 2004).

3.3.2 Process software modelling

The functional behavior of blockchain-oriented software requirements can be described using the Business Process Model and Notation (BPMN), a graphical representation of a business process. BPMN is controlled by OMG and represents a flow-oriented representation of software (Rocha & Ducasse, 2018).

It is aimed to be understandable by all business users, from analysts that create the initial drafts of the processes to the technical developers. BPMN consists of Collaboration diagrams, Process diagrams and Choreography diagrams, and provides a simple means of communicating process information (OMG, 2013).

3.3.3 Data software modelling

In this paper, we focus on the data perspective of software modelling for blockchain-based solutions, including smart contracts and dApps.

There are more different known data models. First, there were three major data models (network, relational, entity set). In 1976, Chen provided a generalized Entity-Relationship (ER) model. It is identified by its graphical representation - ER diagram. It is known as the most natural representation of software, and can be extended for object-oriented software engineering (Chen, 1976; Gorman & Choobineh, 1990). Since smart contracts are similar to conventional classes in object-oriented programming, data in Blockchain-Oriented Software (BOS) can be specified using an ER model for the conceptual and logical design (Rocha & Ducasse, 2018). As Gorman introduced, there are some analogies between the object-oriented programming (OOP) and ER models. These identified analogies are presented in Table 1 (Gorman & Choobineh, 1990).

OOP	ER
class	entity set
instance	entity
variables	attributes
methods	none
object ”visibility”	relationships
sending & receiving objects	roles

Table 1: Analogies between object oriented programming and ER model (Gorman & Choobineh, 1990).

Identified analogies were also used for modelling the data perspective of smart contract in this paper.

4 Data modelling for BOSE

In this section, we focus on the data perspective of BOSE due to the lack of contributions in the field, which we see as a shortcoming. The fact that BOSE is focused mainly around smart contracts, which per-se are stored on the data layer and executed as relational database based functions and/or procedure, calsl for a more data modelling technique. Such a modelling technique, if presented correctly, would emphasize the data perspective of decentralized applications, and outline the different database topology clearly.

In this sense we propose an extension of the de facto Standard for data modelling i.e., the Entity-Relationship Model, which would be modified for BOSE.

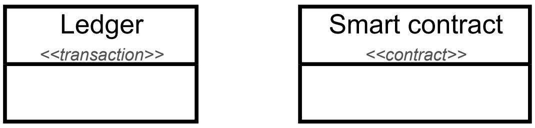

Figure 1: Proposed ER model extension with stereotypes.

4.1 Proposed ER Model extension

Our proposed extension of the Entity-Relationship Model for BOSE contains stereotypes for specific blockchain elements, similar to the proposed UML diagram extension by Marchesi, Marchesi and Tonelli (Marchesi et al., 2020).

The most common blockchain specific element is a smart contract. Since a smart contract is very similar to a class in OOP, we can model it as an entity, where we propose an additional stereotype for distinction between entities in traditional databases and blockchain smart contracts.

Structs in smart contracts can be modeled as attribute type inside a smart contract entity, since they cannot be used outside of the smart contract context.

Variables in smart contracts are modeled as attributes, like in OOP modelling.

Mapping and address are data types in blockchain applications, and can therefore be represented as attribute types.

Ledger is specific element containing all transactions on the blockchain network. Since it represents a set of transactions, it can be modeled as an entity. For distinction between other types of entities, we propose usage of a specific ”transaction” stereotype.

Functions, modifiers and events are structural elements which do not hold any data, so they are not being modeled using an ER diagram.

The proposed entities, with stereotypes, are represented in Figure 1. The proposed modelling technique was validated using the example iPOT application in the following sections.

4.2 Application sample

In order to represent and describe the proposed modelling attributes better, we present these on a case that is built around a real world application, which incorporates blockchain related elements. The application sample covers the support for integrated token payments as part of a comprehensive smart city solution, called iPOT. It contains users and dedicated token (ERC 20) purchases, together with their respective payment types (e.g., paypal), which are all stored in a traditional relational database. On the other side, there are vendors, who provide different services for a specific token price. Users may purchase iPOT tokens, which are later used for the consumption of services. The tokens are minted by a smart contract, which also covers the management and transactions of those. Consumption of services are all stored on the blockchain ledger as transactions to iPOT tokens. The whole history of the blockchain network can be tracked in the ledger log.

The iPOT example application was used for our solution validation, because it represents a real-world application, which incorporates both blockchain and off-chain elements, which can lead to ambiguity if using the existing ER modelling technique.

4.3 Conceptual data model

As a first step in the data modelling approach, we cover the conceptual model.

Using the ER diagram notation, we have modeled our application sample as shown in Figure 2. There are seven entities: Entities User, Token purchase, Payment type, Service and Vendor can be implemented as Tables in a relational database, while entities iPOT token and Ledger log are Blockchain-based entities.

The implementation of blockchain entities differs from the off-chain entities. Consequently, it makes sense to introduce a different notation for blockchain entities for developers to understand the requirements of the project in detail. Therefore, we have added **stereotypes** for a blockchain transaction log and smart contracts in our modified diagram, shown in Figure 2. From that diagram, it is evident that the iPOT token and Ledger are blockchain entities. These entities are marked with stereotypical annotations in our proposed diagram for better visual distinction. Foreign keys are marked with underlined notation, where blockchain addresses are treated as such.

Most of the blockchain software solutions can use a ledger of all transactions, which may be used to verify the history of all interactions with the software. Each blockchain-oriented software may contain one or more smart contract entities, where each smart contract may have one or more instances, according to the implementation logic of each smart contract.

4.4 Logical data model

After defining the high level conceptual data model, the next step is transforming it into the logical data model, which adds additional technical information to the conceptual model, as well as solves possible many-to-many relationships etc. The logical model can be modeled with the extended ER diagram or in a textual format.

Off-chain entities, which may be implemented using a relational database, are modeled using the ER model specifications and data types, as seen in Figure 3. ID is an integer with a primary key role in all off-chain entities. Since a blockchain address consists of 40 hexadecimal characters with 0x prefix, it is presented as an array of 42 characters. Other attributes were assigned data types according to their roles. A blockchain address may work as a foreign key in some situations, therefore, it is underlined.

As marked in Figure 2, an iPOT token is a smart contract containing properties, owner and points. Both of these properties are of data types that are not stated in the ER specification. Therefore, additional data types were used for this logical model design, as shown in Figure 3. An owner is a type of address, which is specific for smart contracts and contains additional members as stated in section 3.2.2. Points are assigned data type mapping, which maps from the data type address to an unsigned integer. Mapping is another specificity of Solidity programming language, which cannot be seen elsewhere.

Furthermore, an entity Ledger is an independent entity, which logs all transactions on the blockchain network. The ledger doesn't need to be implemented, but can be used in any other blockchain entity for obtaining a log of activities on the network.

5 Discussion

The proposed conceptual and logical ER model extensions for the purpose of supporting BOSE present the data structure and requirements of the proposed software. For conceptual design, there are some limitations in the existing specifications of the ER model. Contrary to (Rocha & Ducasse, 2018), we have proposed a stereotype solution to separate blockchain entities from off-chain entities. We propose stereotypes for smart contracts and other blockchain entities, as shown in Figure 3.

Furthermore, we can propose a Table of analogies between BOSE and classical data (ER) model attributes. The analogies are shown in Table 2. There is uncertainty about struct, which may be represented as data type or entity, but since each struct must be part of a smart contract, we have decided to use attribute type as the representation. Functions, modifiers and events are not applied on a data-oriented perspective, since they represent the functionality of smart contracts and not the data structure itself, thus, they are covered in the structure perspective.

BOS	ER
smart contract	entity with stereotype
struct	attribute type
variables	attributes
function	*n/a*
modifier	*n/a*
mapping	attribute type
address	attribute type
ledger	entity with stereotype
event	*n/a*

Table 2: Proposed analogies between blockchain oriented software and the ER model.

Logical data model design was appropriate with additional data types, specific for smart contracts in Solidity programming language.

We have validated our proposed model on a real-world case with a specified iPOT application.

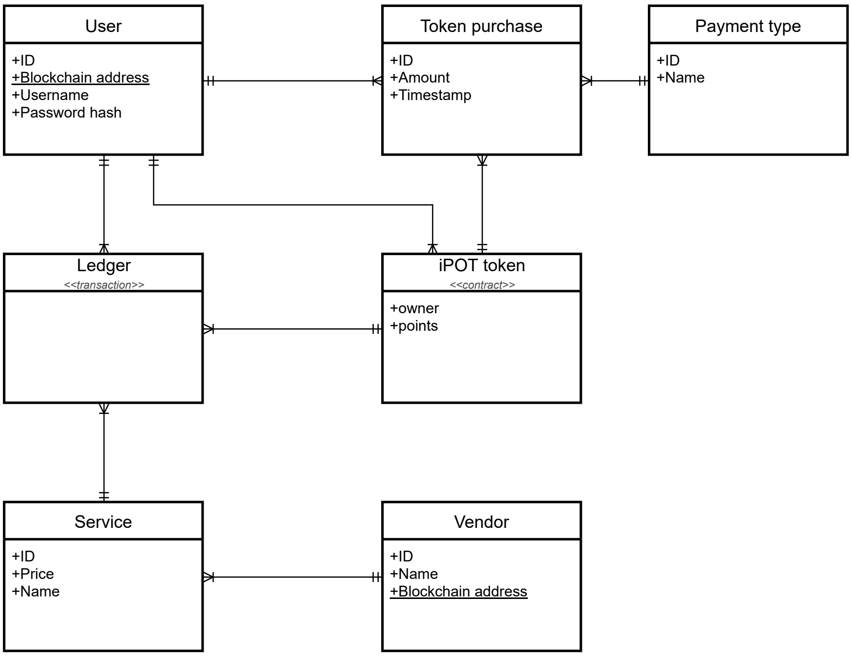

Figure 2: Conceptual data model with modified ER model.

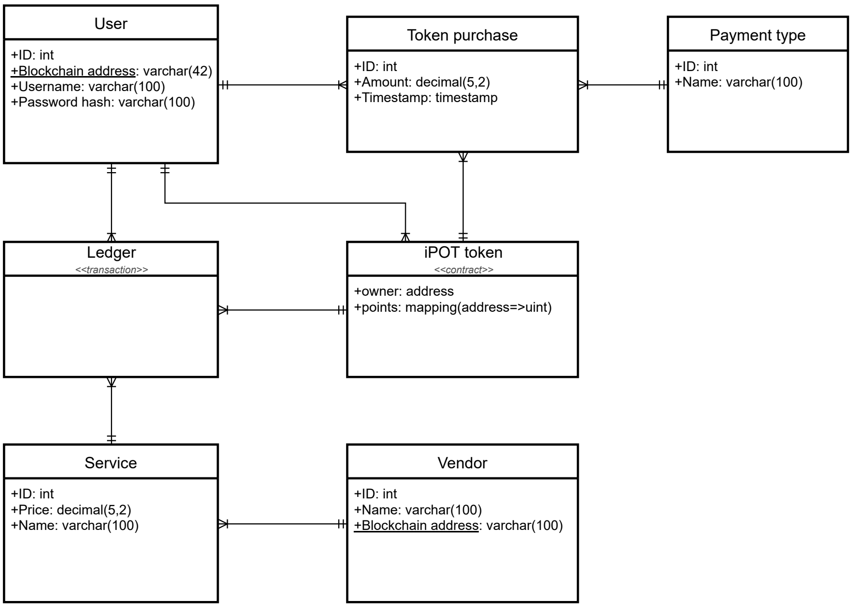

Figure 3: Proposed logical data model design.

6 Conclusion

Existing literature on blockchain-oriented software modelling was analysed in the paper. We focused on the data perspective of modelling, which was not yet addressed in the literature. In order to overcome this shortcoming, we proposed an extension of the traditional ER modelling technique and its ER diagram in order to support the blockchain-based aspects.

Our proposed solution was validated on a real-world case of a blockchain-oriented software based on tangible tokens containing smart contracts and more blockchain-specific entities.

In the future, this model will be evaluated on case studies and in cooperation with experts in the area of blockchain software engineering. There is additional space for research in structure software modelling, which may help software engineers with the design of the behavior of blockchain-based software. Another additional field of research is Data Software modelling for multiple blockchain networks, since our model only applies to a single network.

7 Acknowledgments

This work was supported in part by the Slovenian Research Agency (Research Core Funding No. P2-0057).

References

Aleksieva, V., Valchanov, H., & Huliyan, A. (2020, June). Implementation of smart-contract, based on hyperledger fabric blockchain. In *2020 21st international symposium on electrical apparatus technologies (siela)* (p. 1-4). doi: 10.1109/SIELA49118.2020.9167043

Balaji, S., & Murugaiyan, M. S. (2012). Waterfall vs. v-model vs. agile: A comparative study on sdlc. *International Journal of Information Technology and Business Management*, *2*(1), 26–30.

Beck, K., Beedle, M. A., Bennekum, A. V., Cockburn, A., Cunningham, W., Fowler, M., ... Thomas, D. A. (2013). Manifesto for agile software development..

Chen, P. P.-S. (1976, March). The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.*, *1*(1), 9–36. Retrieved from `https://doi.org/10.1145/320434.320440` doi: 10.1145/320434.320440

Cheng, J.-C., Lee, N.-Y., Chi, C., & Chen, Y.-H. (2018, April). Blockchain and smart contract for digital certificate. In *2018 ieee international conference on applied system invention (icasi)* (p. 1046-1051). doi: 10.1109/ICASI.2018.8394455

Dannen, C. (2017). *Introducing ethereum and solidity* (Vol. 318). Springer.

Gomaa, H. (2011). *Software modeling and design: Uml, use cases, patterns, and software architectures.* Cambridge University Press.

Gorman, K., & Choobineh, J. (1990, Jan). An overview of the object-oriented entity-relationship model (ooerm). In *Twenty-third annual hawaii international conference on system sciences* (Vol. 3, p. 336-345 vol.3). doi: 10.1109/HICSS.1990.205364

Marchesi, L., Marchesi, M., & Tonelli, R. (2020). Abcde –agile block chain dapp engineering. *Blockchain: Research and Applications*, *1*(1), 100002. Retrieved from `https://www.sciencedirect.com/science/article/pii/S2096720920300026` doi: https://doi.org/10.1016/j.bcra.2020.100002

Nakamoto, S. (2009, 03). Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com.*

OMG. (2013, December). *Business Process Model and Notation (BPMN), Version 2.0.2.* Retrieved from `http://www.omg.org/spec/BPMN/2.0.2`

Porru, S., Pinna, A., Marchesi, M., & Tonelli, R. (2017, May). Blockchain-oriented software engineering: Challenges and new directions. In *2017 ieee/acm 39th international conference on software engineering companion (icse-c)* (p. 169-171). doi: 10.1109/ICSE-C.2017.142

Rocha, H., & Ducasse, S. (2018, May). Preliminary steps towards modeling blockchain oriented software. In *2018 ieee/acm 1st international workshop on emerging trends in software engineering for blockchain (wetseb)* (p. 52-57).

Rumbaugh, J., Jacobson, I., & Booch, G. (2004). *Unified modeling language reference manual, the (2nd edition).* Pearson Higher Education.

Wohrer, M., & Zdun, U. (2018, March). Smart contracts: security patterns in the ethereum ecosystem and solidity. In *2018 international workshop on blockchain oriented software engineering (iwbose)* (p. 2-8). doi: 10.1109/IWBOSE.2018.8327565

Wood, G. (n.d.). Ethereum: A secure decentralised generalised transaction ledger.