A COMPARISON OF APPROACHES FOR VISUALIZING BLOCKCHAINS AND SMART CONTRACTS

Felix Härer / Hans-Georg Fill

Research and Teaching Assistant, University of Fribourg, Digitalization and Information Systems Group
Bd de Pérolles 90, 1700 Fribourg, CH
felix.haerer@unifr.ch

Full Professor, University of Fribourg, Digitalization and Information Systems Group
Bd de Pérolles 90, 1700 Fribourg, CH
hans-georg.fill@unifr.ch

Keywords: ***Legal Visualization, Blockchains, Smart Contracts***

Abstract: *The use of blockchains and smart contracts is currently explored in various fields of science and engineering due to their potential of radically changing the ways of doing business and the assumed elimination of traditional legal entities. Thereby, the complexity of the underlying technical relationships and mechanisms typically hampers the understanding by non-technical experts. In this paper we review approaches for visualizing blockchains and smart contracts. The investigation focuses on design and analysis approaches, concluding with requirements for a visual modelling language.*

1. Introduction

While any individual or other legal entity, or software, may engage in blockchain transactions and smart contract applications, the legal implications are typically not well understood. An analysis of the involved contract law and a design of appropriate legal contracts demand a high degree of expertise in the field of law and the underlying technology. Therefore, the analysis and design process usually depends on programmers with little understanding of the legal foundations they operate on. For involving non-technical domain experts in the analysis and design of blockchains and smart contracts, we review in the following according information visualization and modelling methods (Fill, 2006). While traditional contract visualization approaches focus on innovative ways of expressing contracts (Haapio et al., 2018), we review here visualization approaches and practical tools for the purposes of providing technical insight and for the analysis and design of blockchains and smart contracts. After reviewing and discussing these approaches, this paper concludes with requirements for a domain specific modelling language, with the aim to facilitate domain aspects of contract visualization and technical aspects of smart contract design.

The remainder of this paper is structured as follows. Section 2 introduces blockchain technologies with a conceptual model. Section 3 outlines the classification framework and reviews individual approaches, followed by a discussion in Section 4. Based on the results, we outline future research in Section 5 by proposing requirements for an integrated visual modelling language. Section 6 concludes our investigation.

2. Blockchain Foundations

The term blockchain refers to a technology based on a specific data structure of linked blocks together with related consensus algorithms for the verifiably consistent storage of transaction-based data among distributed participants. Today, blockchains function as distributed ledgers for storing transaction histories, as well as smart contract platforms for the autonomous execution of program code tied to transactions. Applications based on distributed ledgers include the transfer of money, assets, securities or any kind of identifiable token by means of transactions, while smart contracts implement decentralized applications for concluding agreements among decentralized participants not necessarily known to each other a priori c.f. (Härer, 2018).

For the purpose of analysis and design from the point-of-view of data, we derived a conceptual data model for the description of distributed ledger and smart contract systems. Figure 1 shows the model together with a framework for analysis and design. For describing a blockchain, the model uses terminology common to most blockchains as it refers to the actual underlying technical components cf. (Fill / Härer, 2018). It resembles blockchain platforms such as Bitcoin and Ethereum.

Regarding the conceptual model, a *Block* is linked to one predecessor by the inclusion of a hash value, summarizing the content of the preceding block. Participants of the system use identifiers in the form of *Address*, each derived from a public-private key pair. A *Transaction* transfers *Value* or *Data* from sender addresses to receiver addresses, where a *Value*, e.g. electronic cash, is a sum of outputs of previous transactions and *Data* either transfers *Contract Code* for the *creation* of a smart contract or executes a previously created contract by an *invocation*. An execution runs *Contract Code* in the execution environment of a blockchain, e.g. the Ethereum Virtual Machine (EVM). Note that in a smart contract blockchain, a *Block* also contains *State* for storing *Accounts*, specifically, *Externally Owned Accounts* accessible by participants or *Contract Accounts* accessible by software through *Contract Code.* Beside the actual code, data resulting from the execution can be stored on the blockchain and read in subsequent executions by assigning it to a *State Variable*. The *Contract Code* itself is a sequence of *Instructions* and *Data-Values*, where instructions perform calculations, functional operations and execution control on the data.

Figure 1: Classification of Visualization Applications for Describing Each Class in Terms of Components of Distributed Ledger or Smart Contract Blockchains

A smart contract blockchain can be understood as a universal computer, akin to a Zuse or Von Neumann architecture. Note that data and instructions may not necessarily be separated. The creation, management and invocation of the components are not shown here in the form of a data model. They are carried out by the consensus algorithm of a blockchain on the infrastructure of a peer-to-peer network.

3. Classification of Visualization Approaches

For classifying the visualization approaches, we refer to the framework shown in Figure 1 along with the contained data model. Classes of visualization applications in the top layer represent existing approaches we review in the following sections. According to the relations marked by (A) and (D), each class of applications is described in terms of the data model components of distributed ledger or smart contract blockchains, where the components represent units of analysis (A) or design (D). For each class, we describe (a.) the analysed or designed aspects and (b.) the visualization as a way for aiding the understanding of the subject matter.

3.1. Design and Distributed Ledger Blockchains

Firstly, approaches on the design of artefacts for distributed ledger systems are classified. This category is relevant for designing state and behaviour of blockchain applications, software connectors and client software.

Model-Driven Design of Blockchchain Applications involves visual models as the primary artefact in the development process of blockchain applications. Approaches adopting model-driven development and domain specific languages argue for better comprehension by specifying applications in terms of a specific domain cf. (Karagiannis et al., 2016). Applications here do not necessarily rely on smart contracts, but may allow for customizations of permissioned blockchains. An example is the *Hyperledger Fabric* blockchain, providing a *Composer* with table-based visualizations of textual models (Gaur et al., 2018), allowing to generate user interfaces as in Figure 2 (a.). The design by a model includes (1.) participant IDs (Address) and custom attributes, (2.) assets as domain-specific objects of value, and (3.) transactions for transferring assets among participants.

Modelling of Blockchchain-based Software Applications concerns the intersection of traditional software systems and blockchains, storing any data of a system on a distributed ledger. Data may include purchase orders, service contracts, or any data. Mostly the creation of addresses or transactions originating from the software side is relevant. One visual design technique are flow-based visual programming languages, such as *Node-RED* (Hyperledger, 2018) shown in Figure 2 (b.). Here, a *Trader* participant is created as specified in a JSON file. Other techniques visualize application behaviour, e.g. extensions of *Petri-Nets* (Norta, 2015). *Petri-Nets* visually and formally describe system states and transitions, i.e. transactions of blockchains may be utilized to transfer from one state to another. Figure 2 (c.) partly shows a non-interactive description of a contract.

Wallet User-Interface-Design concerns client software as the human interface of blockchain systems, however, visualization techniques could only be identified for wallet applications. Wallets (a.) create and manage a participant's addresses and (b.) create and broadcast transactions. Transaction creation visualizations and time-series visualizations of market data[1] for virtual currency, or «cryptocurrency», can be found in practice.

Figure 2: Examples of Distributed Ledger Design Approaches

[1] Exodus, https://steemit.com/@exodus/exodus-1-59-0-it-s-time-for-a-stellar-release (accessed on 29 October 2018).

3.2. Analysis and Distributed Ledger Blockchains

For the analysis of distributed ledger blockchains, various theoretical and practically implemented visualization techniques exist. Approaches in this class mostly visualize structural components given by the framework.

Chart-based Statistical Evaluation techniques aid the understanding of categorical data of distributed ledgers. The transparent nature of permissionless blockchains provides open data access for analysing operational parameters. Tools such as OXT[2] and Bitcoin Visuals[3] represent aggregated data in various forms such as scatter plots, line and bar graphs, e.g. for distributions of money circulation, block size or cost. A use case is the investigation of adverse behaviour, e.g. by adjacency matrices of money flows for detecting spam transactions (McGinn et al., 2018).

Block-Transaction Visualization concerns the analysis of transactions included in blocks over time, allowing for investigations into all transactions carried out by a blockchain system at a specific point in time. Block explorer tools provide such a view of transactions on specific blocks by allowing queries by block number or other identifying attributes, such as the block hash value. A discussion of an analytical process is provided by (Kuzuno / Karam, 2017). Tools such as Blockchair[4] and Blockcypher[5] are web-based implementations. As an example, Figure 3 (a.) shows transactions of block 554216 of the Bitcoin blockchain.

Address-Transaction Visualization is a view of all transactions sent or received by a specific address over time, regardless of the block. Such a view may also be found in the block explorers mentioned. An analysis into the activity of specific addresses is the typical use case, for the investigation of benevolent or malicious participants. A visual exploration as discussed in (Kinkeldey et al., 2017) allows e.g. for showing activity of a group of related addresses, possibly as part of a wallet for an identity.

Transaction Graphs are the most common representation to analyse the flow of value across any number of addresses over time. A graph is constructed by nodes in the form of addresses, or groups of related addresses, connected by directed edges in the form of transactions, from sender to receiver addresses. Figure 3 (b.) shows a colour-coded graph (Möser, 2013) for analysing anonymity and money laundering. In addition, individual values of one transaction acting as inputs on the sender side and as outputs on the receiver side, might be shown (Battista et al., 2015) as in Figure 3 (c.). Other representations exist, e.g. as *Petri-Nets* (Pinna et al., 2018).

Figure 3: Examples of Distributed Ledger Analysis Approaches

Network Analysis takes the transaction graph approach as a basis and applies methods from network theory, social network analysis and artificial intelligence. Following individual flows of money is complemented by identifying related participants and communities, e.g. reverting to centrality measurements, cliques or advanced forms of clustering and unsupervised machine learning (Kalodner et al., 2017), (Harlev et al., 2018).

[2] Oxt, https://oxt.me/charts (accessed on 14 December 2018).

[3] Bitcoin Visuals, https://bitcoinvisuals.com/ (accessed on 14 December 2018).

[4] Blockchair, https://blockchair.com/ (accessed on 14 December 2018).

[5] Blockcypher, Bitcoin Block 554,216, https://live.blockcypher.com/ (accessed on 17 December 2018).

Network Node Map relates client software instances operating a blockchain's peer-to-peer-network to spatial dimensions. An application may be to show the degree of decentralization of a blockchain system in terms of number of nodes and geographic location. An example for the bitcoin network is the tool Bitnodes[6].

3.3. Design and Smart Contract Blockchains

The classification of visual approaches for designing smart contracts and related components distinguishes visualizations of procedural programming, declarative logic programming and modelling languages.

Visual Programming Languages allow for the specification of smart contracts by the graphical representations of (1.) model elements representing instructions and data-values as well as (2.) model relations for defining the control flow between elements at least in the form of sequences and branching dependent on specifiable conditions, e.g. for loops and conditional execution. A mapping to instructions of a smart contract programming language defines the generation of contract code. The result is either human-readable source code of a smart contract programming language like Solidity, or byte code interpretable by the node software of the targeted blockchain. In the former case, a compilation to byte code is required before execution. One form of visual programming languages are the flow-based languages already shown in Section 3.1. Another form are block-structured representations. This form implies a structuring of a smart contract into hierarchical blocks of code. In the example in Figure 4 (a.), the contract code for a sales contract is shown in the Ether-Scripter[7] tool and explained below. The Serpent or LLL source code for Ethereum can be seen online for this example.

Figure 4: Examples of Smart Contract Design and Analysis Approaches

The contract for selling a website executes on the Ethereum blockchain. After the creation of the contract by a data transaction containing byte code, a *BUYER* and a *SELLER* are stored along with their addresses in state variables on the blockchain. A value holding a date and a time is assigned to another state variable for setting a *DEADLINE* as the latest possible point for the sale to take place. On a subsequent invocation of the contract by a transaction, the sale is carried out if (1.) at least 5000 units of *ether* have been payed to the contract account and (2.) the transaction has been stored in a block before or exactly at the *DEADLINE*. The sale is concluded by recording the *BUYER* as *WEBSITE_ADMIN* on the blockchain as well as transferring the balance of the contract account to the *SELLER*.

Declarative Logic for Smart Contracts is another specification approach, however, based on the paradigms of declarative and logic programming. A contract's logic is represented by formalized rules which can be mapped to contract code in the form of source code or byte code. Logic-based specifications have been suggested (Governatori et al., 2018), benefiting of interoperability among rule markup languages, such as LegalRuleML. Another suggestion are contract definitions by reciprocal commitments (de Kruijff / Weigand, 2018). While approaches here are promising, domain-related visual representations are not part of them.

Domain Specific Modelling Languages allow for the design of a model in terms of a specific domain. Akin to the textual variant of it, mentioned in Section 3.1, models are expressed in the language of domain experts

[6] Bitnodes, https://bitnodes.earn.com/nodes/network-map/ (accessed on 14 December 2018).

[7] EtherScripter, http://etherscripter.com (accessed on 29 October 2018).

and are, here, translated to smart contract code. Approaches could be identified for the generation of Ethereum contract code from (1.) rules specified in the Decision Model and Notation (DMN) (Haarmann et al., 2018), (2.) processes specified by Business Process Model and Notation (BPMN) e.g. (Tran et al., 2018), and (3.) ontologies and semantic rules (Choudhury et al., 2018). While DMN rules could in theory be adapted as a basis for deriving smart contracts from legal contracts, no specific approaches for this purpose could be found.

3.4. Analysis and Smart Contract Blockchains

Visual approaches for the analysis of smart contracts rely on statistics, the analysis of transactions of contracts and advanced methods for investigating smart contracts by decompilation and control flow analysis.

Chart-based Statistical Evaluation techniques aid the understanding of categorical data of smart contract blockchains. Similar to the class in Section 3.1, operational parameters can be found. In addition, statistics of aggregated smart contract creations and invocations are visualized, e.g. in tools such as Ethstats[8].

Contract-Transaction-Analysis concerns the analysis of transactions sent to a receiver address of a specific contract account. The transactions recorded show (1.) the creation of a contract at a specific block, and (2.) each invocation at a specific block, executing the contract. As part of the block, (approximate) timestamps are available. An example of an implementation can be found in a block explorer[9].

Smart Contract Decompiler provide insight into the source code of a smart contract, given the byte code. This is a challenging task, since byte code is a non-human-readable specification for an execution environment, e.g. the EVM as a stack machine in Ethereum. In this representation, instructions are encoded for a particular environment by specific values of data, known as «opcodes», and values of data from state variables are not stored under their original names. Given these limitations, a logical source code representation agnostic to semantics might be retrievable. Ethervm[10] is an example for generating Ethereum solidity source code.

Control Flow Analysis determines the execution behaviour from the byte code of a smart contract. The analysis is similar to a decompilation, however, for a given smart contract the result shows possible sequences of executed instructions. Figure 4 (b.) visualizes executable sequences (Norvill et al., 2018), where a node in the graph can be reached by executing opcodes shown on the left-hand side. A *jump* instruction continues the execution at a location shown on the stack at the right-hand side. Execution states visited in previous blocks are indicated in blue. The graph can be shown for different blocks, allowing an analysis over time.

4. Discussion

The manifold ways of visualizing the analysis and design of the components discussed may, to an extent, help in providing an insight into distributed ledgers and smart contracts. While the visual representation of components can by itself be beneficial for the facilitation of design and analysis tasks, an intuitive understanding established by the visual concepts themselves cannot be assumed, given the examples known and reviewed.

Fundamentally, the visual design and analysis approaches found for blockchain platforms do allow for a design of legal contracts, primarily for cases where the transfer of ownership is the subject matter. The design may specify the parties involved, the object of the contract and the conditions for the conclusion of a contract by means of blockchain transactions. While the specification is visualized when applying model-based approaches, technical knowledge about a chosen blockchain platform and a fundamental understanding of the involved visual programming techniques are still pre-requisites. One reason is that visual models are presented on the same level of abstraction as technical models. In model-based design approaches for blockchain systems such as Hyperledger, as well as in visual programming languages for the design of smart contracts, little abstraction is gained by a one-to-one-substitution of each textual command for a visual object. Similarly, the visual analysis approaches operate on the level of technical concepts. For analysing the conditions causing the conclusion of a smart contract, and for an analysis of the blocks, transactions and addresses affected by the conclusion, block explorer tools and advanced approaches such as control flow analysis are not designed to provide higher level abstractions. Visualization by itself is, thus, insufficient. While the reviewed design approaches might be appropriate for technical development processes, and while the analysis approaches might provide insight when conducting targeted investigations, their broader applicability in domains outside

[8] Ethstats, https://ethstats.net/ (accessed on 14 December 2018).

[9] Bloxy, https://bloxy.info/ (accessed on 14 December 2018).

[10] Ethervm, https://ethervm.io/decompile (accessed on 14 December 2018).

of technical fields is limited. At the same time, fields such as the legal domain are directly concerned with smart contracts and distributed ledgers when it comes to actual contracts and transactions.

For this reason, we propose further research efforts towards the direction of domain specific analysis and design methods. This leads us to the following proposition of requirements for designing an integrated visual modelling language.

5. Requirements for an Integrated Visual Modelling Language

The modelling language must address the design and analysis of distributed ledger and smart contract artefacts according to the following requirements.

R1: Analysis or design must be expressible in the language of the domain, for it to be used by domain specialists. Specifically, any domain concept, relevant for design and analysis, must be available as a modular element for the construction of an analysis or a design. At least, the assessment of transactions and smart contracts, for judging their implications, as well as the construction of transactions and smart contracts must be covered.

R2: Domain concepts must establish their meaning in the context of the analysis or the design by themselves, using visualization techniques. For example, when designing a smart contract, the semantics of the domain concepts modelled must be clear in the context of any of the affected parts of the contract. Methods for realization may combine techniques such as flow-based and block-structured visualizations with domain concepts.

R3: For the design and analysis as part of complex systems, such as distributed ledger and smart contract blockchains, the level of abstraction must match the level of the domain. For interactions with such a system in any specific domain, the level of abstraction in all interactions must be the same as the level prevalent in the domain. In order to conduct a legal analysis of a smart contract, e.g. assessing it as a legal contract, the level of abstraction is at the level of legal concepts and their language. Note that in other domains, levels of abstraction differ vastly, e.g. it is at the level of control flow analysis for performing security audits.

6. Conclusion

For visualization approaches, we introduce a classification in terms of components of distributed ledger and smart contract systems. By visualization, approaches and tools provide insights mostly to technical users. For legal analysis and design, visualizations are not sufficient by themselves. With the aim of facilitating design and analysis beyond technical fields, we propose a language using abstractions of a domain. In future research, we plan to investigate (1.) the technical realization, (2.) the representation of domain requirements, and (3.) according visualizations for concepts and context, for domain specialists to act as designers and analysts.

7. References

Battista, Giuseppe / Donato, Valentino / Patrignani, Maurizio / Tamassia, Roberto, BitConeView: Visualization of Flows in the Bitcoin Transaction Graph, IEEE Symposium on Visualization for Cyber Security, Chicago 2015.

Choudhury, Olivia / Rudolph, Nolan / Sylla, Issa / Das, Amar, Auto-Generation of Smart Contracts from Domain-Specific Ontologies and Semantic Rules, IEEE Blockchain Conference, Halifax 2018.

Fill, Hans-Georg, Visualisation for Semantic Information Systems, Gabler/Springer, 2006.

Fill, Hans-Georg / Härer, Felix, Knowledge Blockchains: Applying Blockchain Technologies to Enterprise Modeling. Proceedings of the 51st Hawaii International Conference on System Sciences (HICSS), 2018, pp. 4045–4054.

Gaur, Nitin / Desrosiers, Luc / Novotny, Petr / Ramakrishna, Venkatraman / O'Dowd Anthony / Baset, Salman, Hands-On Blockchain with Hyperledger, Packt Publishing, Birmingham 2018.

Governatori, Guido / Idelberger, Florian / Milosevic, Zoran / Riveret, Regis / Sartor, Giovanni / Xu, Xiwei, On legal contracts, imperative and declarative smart contracts, and blockchain systems, Artificial Intelligence and Law, volume 26, issue 4, 2018, pp. 377–409.

Haapio, Helena / deRooy, Roberg / Barton, Thomas D., New Contract Genres, in: Schweighofer, Erich / Kummer, Franz / Saarenpää, Ahti / Schafer, Burkhard (Eds.), Data Protection / LegalTech, Proceedings of the 21th International Legal Informatics Symposium IRIS 2018, Editions Weblaw, Bern 2018, pp. 455–460.

Haarmann, Stephan / Batoulis, Kimon / Nikaj, Adriatik / Weske, Matthias, DMN Decision Execution on the Ethereum Blockchain, in: Advanced Information Systems Engineering (CAiSE 2018), Tallinn 2018.

Härer, Felix, Decentralized Process Modeling and Instance Tracking Secured by a Blockchain, European Conference on Information Systems (ECIS 2018), Portsmouth 2018.

Harlev, Mikkel / Sun, Yin Haohua / Langenheldt, Klaus Christian / Mukkamala, Raghava / Vatrapu, Ravi, Breaking Bad: De-Anonymising Entity Types on the Bitcoin Blockchain Using Supervised Machine Learning, Proceedings of the 51st Hawaii International Conference on System Sciences (HICSS), 2018, pp. 3497–3506.

Hyperledger, Node-RED Tutorial for Integrating a Hyperledger Composer Business Network, https://hyper ledger.github.io/composer/v0.16/tutorials/nodered-tutorial (accessed on 14 December 2018).

Kalodner, Harry / Goldfeder, Steven / Chator, Alishah / Möser, Malte / Narayanan, Arvind, BlockSci: Design and applications of a blockchain analysis platform, arXiv:1709.02489 [cs.CR], 2017.

Karagiannis, Dimitris / Mayr, Heinrich C. / Mylopoulos, John, Domain-Specific Conceptual Modeling, Springer International Publishing, 2016.

Kinkeldey, Christoph / Fekete, Jean-Daniel / Isenberg, Petra, BitConduite: Visualizing and Analyzing Activity on the Bitcoin Network, Eurographics Conference on Visualization (EuroVis 2017), Aire-la-Ville (CH) 2017, pp.3.

de Kruijff, Joost / Weigand, Hans, Commitment-Based Smart Contracts Using RuleML, RuleML Conference, 2018, https://www.researchgate.net/publication/326088881 (accessed on 14 December 2018), 2018.

Kuzuno, Hiroki / Karam, Christian, Blockchain explorer: An analytical process and investigation environment, 2017 APWG Symposium on Electronic Crime Research (eCrime 2017), 2017, pp. 9–16.

McGinn, D. / McIlwraith, D. / Guo, Y., Towards open data blockchain analytics: a Bitcoin perspective, Royal Society Open Science, 2018, DOI 10.1098/rsos.180298.

Möser, Malte, Anonymität von Bitcoin Transaktionen, Analyse von Bitcoin Mixern, Münster Bitcoin Conference, Münster 2013, https://www.wi.uni-muenster.de/sites/wi/files/public/department/itsecurity/mbc13/mbc13-moeser-presentation.pdf (accessed on 5 December 2018).

NORTA, ALEX, Creation of Smart-Contracting Collaborations for Decentralized Autonomous Organizations. Proceedings of Business Informatics Research (BIR 2015), 2015, pp. 3–17.

NORVILL, ROBERT / PONTIVEROS, BELTRAN BORJA FIZ / STATE, RADU / CULLEN, ANDREA, Visual emulation for Ethereum's virtual machine, in: IEEE/IFIP Network Operations and Management Symposium (NOMS), Taipei 2018, pp. 1–4.

PINNA, ANDREA / TONELLI, ROBERTO / ORRÙ, MATTEO / MARCHESI, MICHELE, A Petri Nets Model for Blockchain Analysis, Computer Journal, volume 61, issue 9, 2018, pp. 1347–1388.

TRAN, AN / LU, QINGHUA / WEBER, INGO, Lorikeet: A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management, Business Process Management (BPM 2018), Sydney 2018.