POLITECNICO DI TORINO
Repository ISTITUZIONALE

Smart Contract Visualization: Solutions and Challenges

Original
Smart Contract Visualization: Solutions and Challenges / Napoli, E.A., Spada, I., Romani, N., Gatteschi, V., Schifanella, C.. - 4105:(2025). (DLT 2025: 7th Distributed Ledger Technologies Workshop Pizzo, VV (ITA) June 12-14 2025).

Availability:
This version is available at: 11583/3004390 since: 2025-10-23T09:09:18Z

Publisher:
CEUR

Published
DOI:

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

05 November 2026

Smart Contract Visualization: Solutions and Challenges

Emanuele Antonio Napoli[1,*], Ivan Spada[2], Noemi Romani[1], Valentina Gatteschi[1] and Claudio Schifanella[2]

[1]Politecnico di Torino, Corso Duca degli Abruzzi, 24, Turin, 10129, Italy
[2]Università degli Studi di Torino, Via Verdi, 8, Turin, 10124, Italy

Abstract

The advent of distributed ledger technology across various sectors highlighted its potential to revolutionize conventional processes and systems. In particular, smart contracts emerged as disruptive innovation, automating legal contract clauses between parties. However, smart contract development is not an easy task and raises nontrivial challenges to non-programmers since they need to have specialized programming skills and deep understanding of blockchain technology. These challenges reduce user engagement and increase the possibility of introducing bugs and vulnerabilities that can lead to huge financial loss. To address these complexities, visual programming has emerged as the main solution among researchers since it is a well-proven methodology to improve and lower the learning curve of a new programming language and create a working piece of code in a user-friendly way. This paper aims to assess the visual formalism and representation that have been used in academia to represent smart contracts in an easy-to-understand manner. The findings show a clear evolution in visualization approaches, with Business Process Model Notation (BPMN) emerging as the dominant methodology (35%) in recent years. Behavioral visualizations (52.5%) have increasingly replaced structural representations (37.5%) as the field matures, whereas evolution-focused approaches remain underexplored (10%). Despite balanced accessibility requirements across user expertise levels, only 20% of studies incorporate formal UX testing, suggesting significant opportunities to improve user-centered design in the visualization of smart contracts.

Keywords

Smart Contract, Visualization, Graphical Representation, Smart Contract Visualization, Blockly, UML, BPMN

1. Introduction

Distributed ledger technology represents a major shift in data management field such as transactions, or file documents, since the information is shared across decentralized networks instead of persisting on one (or few) centralized repositories. Since every peer in the network has all the data, the distributed ledger enables levels of transparency, immutability, and consensus-driven validation without relying on third parties. Blockchain technology is an example of distributed ledger which uses cryptography to ensure that once the data are committed together, they remain unmodifiable or tamper-resistant, thereby making a chronological record that all participants can independently verify. Due to the distributed nature of the system, blockchain is remarkably resistant against single-point of failure and censorship attempts since the data exist simultaneously on multiple computational nodes. These characteristics are enforced by the so-called "blocks" of transactions, the most common data on blockchain, linked together through cryptographic hashing. This particular technology enables the execution of smart contracts, pieces of code that run on top of the blockchain which model business logic into code that can be triggered by user transactions. Since the data on the blockchain are publicly available and cannot be modified once included in a block, smart contracts are extremely reliable because, once the authenticity of the code is verified, anyone can trust it knowing that no one can modify it. However, the immutability properties of blockchain technology, while beneficial for trust, introduce significant challenges. Indeed, deployed smart contracts cannot be patched if bugs or vulnerabilities are detected, potentially leading

DLT2025: 7th Distributed Ledger Technology Workshop, June, 12-14 2025 - Pizzo, Italy
*Corresponding author.
emanuele.napoli@polito.it (E. A. Napoli); ivan.spada@unito.it (I. Spada); noemi.romani@polito.it (N. Romani); valentina.gatteschi@polito.it (V. Gatteschi); claudio.schifanella@unito.it (C. Schifanella)
0009-0006-5221-8070 (E. A. Napoli); 0000-0001-7116-9338 (I. Spada); 0009-0004-3725-8002 (N. Romani); 0000-0001-6075-6430 (V. Gatteschi); 0000-0001-7449-6529 (C. Schifanella)
© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2025-11-24

CEUR-WS.org/Vol-4105/paper10.pdf

to huge financial losses. This has promoted extensive research efforts to develop vulnerability detection methods before deployment [1]. As blockchain technology gained popularity, many newcomers began deploying smart contracts by simply copying code from established entities [2], often without proper verification. This practice emerged because many blockchain enthusiasts lack development expertise, and even those with programming experience face the dual challenge of understanding blockchain's complex principles while simultaneously learning specialized programming languages like Solidity. Visual programming tools have historically proven effective for teaching complex technical concepts by allowing learners to focus on core logic rather than syntax details. These tools can significantly flatten the learning curve by providing intuitive representations of abstract concepts. Despite the evident need for accessible development tools, a comprehensive overview of existing approaches to visual representation of smart contracts is currently lacking. This paper systematically maps efforts to represent smart contracts through visual formalisms, analyzing proposed visual methods to understand their strengths, limitations, and underlying design philosophies. This analysis helps identify which visualization techniques best support different aspects of smart contract development—from initial learning to security analysis to collaborative design. While the blockchain ecosystem continues its rapid growth, the barrier to entry remains substantial due to specialized programming languages and complex execution models. Visual representations could democratize access to this technology while promoting best practices, but first requires understanding what approaches have been attempted, what works effectively, and what gaps persist. This mapping study establishes a foundation for developers and researchers to build enhanced visual tools that make smart contract development more accessible without compromising essential capabilities or security considerations.

The paper is organized as follows. In Section 2 a useful key aspect is provided to introduce the context and rationale behind this work. Section 3 provides the research methodology used to carry out the analysis. The main categories of smart contract visualization identified from the analysis of the selected articles are presented in Section 4 while the mapping and analysis of the selected papers are presented in Section 5. Finally, conclusions are reported in Section 6.

2. Background

Understanding complex software systems requires effective methods to clearly represent how they operate. Visualization approaches in software engineering have been widely adopted to support this need. In parallel, the emergence of smart contracts has introduced new programming paradigms that demand both security and formal correctness. These contracts are typically developed using specialized programming languages such as Solidity, the most widely used language for Ethereum, as well as Vyper and Rust (commonly used on platforms like Solana and NEAR). Although these languages emphasize determinism and transparency, their syntactic and semantic complexity can pose significant challenges for novice developers. The integration of visual programming and visualization techniques into smart contract development offers promising potential to improve accessibility, comprehension, and reliability. The following section provides an overview of both software visualization techniques and the programming languages used to develop smart contracts, in order to clarify the rationale behind the design choices made in proposing a visual programming solution in literature for smart contract development.

2.1. Code Visualization

Three main categories have been identified in general software visualization: structure, evolution, and behavior [3, 4]. Structure visualization focuses on analyzing information such as source code [5], library dependencies [6], and control flow graph [7]. Evolution visualizations focus on code change history to illustrate how a software system evolves over time [8]. Finally, behavior visualization uses data collected from program execution, such as function calls [9], to support tasks like performance optimization [10], and anomaly detection [11]. Existing behavior visualizations often present data as time series [12] or domain-specific event sequences [10]. Visual programming environments primarily emphasize the

visualization of code structure, aiming to abstract away complex textual syntax and enable learners to focus more on programming logic rather than syntactic details. This abstraction is particularly valuable in educational contexts, where novice programmers struggle with the intricacies of traditional text-based programming languages [13]. The visual programming approach has significant implications for both improving programming skills and managing cognitive load. Regarding programming skill improvement, block-based visual programming languages and platforms effectively teach fundamental computer science concepts by facilitating comprehension and algorithm implementation through visual interfaces and immediate program state visibility. Platforms like Scratch and Blockly have proven helpful for learning text-based programming and developing computational thinking [14]. Their visual nature reduces syntax errors and makes programming concepts more accessible to beginners. Research indicates that integrating block programming with interactive physical objects, such as in smart home applications, positively impacts programming skills and student confidence by creating tangible learning artifacts [15]. Recent literature has introduced several notable block-based visual programming platforms. Algot, based on programming by demonstration, improves understanding of complex concepts such as recursion among high school students and improves task performance for university students compared to text-based languages like Python [16]. Similarly, NextBlock – integrated into Moodle – enables educators to create personalized exercises and promote collaborative learning, while adapting activities to specific student needs [14]. Concerning cognitive load management, visual programming offers several burden-reducing mechanisms. A study using eye tracking and galvanic skin response found a lower cognitive load for university students performing simple tasks in Algot compared to Python. However, a subsequent EEG study showed similar levels of cognitive load for recursion-based tasks, despite better performance in Algot [17]. The authors suggest that improved performance with similar cognitive demands indicates Algot's utility for beginners. Block-based programming generally reduces cognitive load by breaking complex constructs into visual blocks and decreasing the information processed simultaneously. The drag-and-drop interface and the prevention of syntax errors create a less frustrating programming environment for novices. Inspired by the principles of "Learnable Programming", Algot maintains the visibility of the continuous state of the program and allows direct interaction, simplifying learning by clarifying the meaning of the program [18]. Similarly, NextBlocks reduces the cognitive load by eliminating syntax memorization requirements, allowing students to focus on programming logic [14].

2.2. Smart Contract Programming Language

Smart contracts are executable rules that can be triggered by the users with transactions. Smart contract programming languages enable developers to create self-executing agreements on blockchain platforms. These languages vary significantly in their programming paradigms, type systems, and state management approaches, reflecting the diverse architectural choices of their respective blockchain platforms. Rust serves as Solana's primary smart contract language, implementing a stateless account-based model where procedures operate on separate data accounts. The Anchor framework helps abstract away some of Rust's complexity in the blockchain context. Aiken, a functional language for Cardano, works with the extended UTXO model, requiring developers to specify transaction validation conditions rather than procedural state changes. Move, embedded in platforms like Aptos, features linear types to ensure that tokens cannot be replicated or lost. PyTEAL provides Python bindings for Algorand's TEAL bytecode, while SmartPy offers meta-programming capabilities for Tezos. Each language makes different trade-offs in terms of expressiveness, safety, and abstraction level. Last but not least, Solidity, developed in 2014, remains the dominant high-level language for Ethereum Virtual Machine (EVM) compatible blockchains including Ethereum, Avalanche C-Chain, and Hedera. Using a JavaScript-like syntax with object-oriented features, Solidity follows an account-based stateful model where contracts act like classes with methods and storage. Despite its accessibility, Solidity has design quirks that can lead to security vulnerabilities in decentralized applications. Most of the works present in the literature target the visual representation of smart contract that are then translated in Solidity since it is the most widely spread smart contract programming language [19].

2.3. Motivations

Despite some relevant work in the literature, there remains a significant need to comprehensively map and analyze visual formalisms for smart contracts. Vieira and Vilain [20] demonstrated that state diagrams can effectively visualize smart contracts, improving comprehension and development accuracy. Their research identified visualization techniques including finite-state machines, UML state machine diagrams, Business Process Model Notation (BPMN), and Petri Nets, with experiments showing that state diagrams significantly reduced comprehension time compared to natural language descriptions alone. Their approach bridged the semantic gap between legal contracts and technical implementations while making smart contracts more accessible to non-technical stakeholders. Furthermore, Curty et al. [21] highlight that the creation of blockchain-based software applications requires considerable technical knowledge, particularly in software design and programming. This is regarded a major barrier to adopting this technology in business and making it accessible to a wider audience. As a solution, low-code and no-code approaches have been proposed that require only little or no programming knowledge to create full-fledged software applications. Their review of academic approaches from the discipline of model-driven engineering, as well as industrial low-code and no-code development platforms for blockchains, includes a content-based, computational analysis of relevant academic papers and the derivation of major topics.

Our work aims to map and analyze the visual formalisms proposed in literature to represent smart contracts in terms of source code, interaction, or state evolution. As demonstrated in Section 2.1, visual approaches can significantly improve the comprehension of source code and reduce the cognitive workload of users. By systematically investigating the various visualization techniques employed across different smart contract platforms and programming languages, this work aims to identify effective representation patterns, uncover gaps in existing approaches, and propose standardized visualization practices that could enhance both educational results and professional development workflows in the smart contract ecosystem.

3. Research methodology

To achieve the objectives of this study, an initial search was performed using the SCOPUS search engine to establish familiarity with the existing literature on the subject matter. SCOPUS includes scientific databases including ACM Digital Library, IEEE Xplore, Elsevier, Wiley, and Springer. The search used multiple strings: (“smart contrac”) AND ((“visualization” OR “representation”) OR (“visual” AND “abstraction”) OR (“visual” AND “representation”) OR (“visual” AND “programming”)).

After the initial collection, 421 papers were screened first by title and then by abstract to determine relevance. For a paper to be included in the final review, it needed to meet the following criteria:

- be written in English;
- be available in full text online;
- offer a clear visual representation of smart contracts in at least one of the following aspects: structure of the source code, business interactions between parties, or evolution of the transition states of the smart contract.

A significant number of papers were excluded because they focused on graph neural networks or merely presented their findings in a visual way rather than offering visual representations of the smart contracts themselves. Articles that did not meet these criteria were excluded from further analysis. Following the collection of initial resources, the search was expanded to include relevant articles resulting from strings “smart contract” AND (“no-code” OR “low-code” OR “bpmn” OR “mda” or “uml”) which led to the selection of 119 articles. A secondary search was conducted through the citations and references identified during the primary search process.

The investigation yielded 40 scholarly articles relevant to the research focus after the screening phase and the elimination of duplicates. Informed conclusions regarding the current state of the field were

derived through in-depth examination of these papers, and potential avenues for future enhancement within this domain were identified.

4. Smart Contract Visualization Approaches

The following section provides a summary of various approaches to visual representation of smart contracts proposed by academia. To facilitate a clearer understanding, the approaches are categorized in Block-based, Model-Driven, and Custom visualization approaches. The proposed visual solutions are presented in Table 1 reporting visual methodology, visual category (as explained in Section 2.1), public availability of the tool, presented use case, target user, and whether the usability of the proposed tool has been tested.

4.1. Block-based Visualization Approach

Several researchers have adapted UML for smart contract visualization. Marchesi et al. [22] extended traditional UML diagrams with specialized formalisms for Solidity constructs and enhanced sequence diagrams with blockchain-specific stereotypes. Garamvolgyi et al. [23] employed UML statecharts to model cyber-physical systems, capturing complex interactions between physical devices and their blockchain representations. Pierro [24] developed Smart-Graph to generate augmented UML class diagrams that include Solidity-specific constructs absent from traditional UML standards. Heckel et al. [25] created a visualization framework using UML-inspired class diagrams enriched with domain-specific stereotypes for DAML templates. Jurgelaitis et al. [26] proposed MDAsmartCD, a comprehensive UML-based framework that spans the entire model-driven architecture lifecycle with blockchain-specific extensions. Ghaffari Saadat [27] presented a unique approach using symbolic attributed graph grammars to visualize smart contracts through typed graphs with color-coded transformation rules.

A significant research trend employs block-based visual programming to make smart contract development accessible. Weingärtner et al. [28] introduced a modular Blockly framework that organizes blocks into types, delivery options, and contractual options. Guida and Daniel [29] developed SolidityEditor with more than 70 custom blocks for Solidity constructs. Mao et al. [30] combined Blockly with neural code generation to produce function templates encapsulated in visual blocks. Merlec et al. [31] presented SmartBuilder, organizing blocks into functional domains while providing natural language summaries. Trestioreanu et al. [32] created Blockly2Hooks for XRP Ledger development with an end-to-end pipeline from visual programming to deployment. Gomez et al. [33] introduced SmaCly with predefined ERC-standard templates and semantic compatibility enforcement. Tsai et al. [34] developed an educational framework extending Blockly for blockchain education among K-12 students.

These visualization methodologies share several key objectives regardless of their underlying formalism. All approaches aim to abstract programming complexity and enforce structural correctness. UML-based approaches excel at modeling system architecture and behavioral semantics, while block-based systems focus on executable code generation and practical development.

Most frameworks provide immediate feedback loops between visual representation and generated code. Several techniques incorporate domain-specific extensions to capture blockchain-specific concepts missing in standard modeling notation. There is also a consistent emphasis on lowering technical barriers for non-experts while preserving the semantic richness required for blockchain applications.

No.	Year	Reference	Visual Approach	Visualization Category	Available Online	Use case	Accessibility Requirements	UX testing
1	2018	[35]	JSON HTML REST Pharo UI	Behavior	Yes	Pool	Basic	No
2	2018	[28]	Blockly	Structure	No	Industrial contracts for machinery	Basic	No
3	2018	[22]	UML	Structure	No	Voting system	Basic	No
4	2018	[23]	UML	Evolution	Yes	Door control mechanism	Intermediate	No
5	2019	[29]	Blockly	Structure	Yes	Train Insurance	Intermediate	No
6	2019	[30]	Blockly	Structure	No	Supply Chain	Basic	No
7	2019	[36]	BPMN	Behavior	Yes	Invoicing Supply chain Incident management Insurance claim	Intermediate	No
8	2020	[27]	UML	Structure	No	Escrow	Advanced	No
9	2020	[37]	Action graph of nodes	Structure	Yes	Voting system	Basic	No
10	2020	[38]	UML	Behavior	Yes	Real Estate Mortage case	Basic	No
11	2021	[24]	UML	Structure	Yes	Solidity Library “Owner”	Advanced	No
12	2021	[20]	UML	Structure	No	Purchase of paper sheets	Basic	6 users
13	2021	[39]	BPMN	Behavior	Yes	Mortgage case	Basic	No
14	2021	[40]	EFSM Model	Evolution	No	Airline delay insurance Betting transactions Auction Baggage insurance Combined contract	Intermediate	No
15	2021	[31]	Blockly	Structure	Yes	Payment system	Basic	No
16	2021	[41]	BPMN	Behavior	No	Payment in construction work	Basic	No
17	2022	[42]	Answer Set Programming (custom)	Structure	No	Real Estate	Advanced	No
18	2022	[43]	Flow-based programming (custom)	Evolution	No	6 smart contract scenarios (Not specified)	Basic	58 students
19	2022	[25]	UML	Structure	No	Voting system	Intermediate	No
20	2022	[44]	DSL	Behavior	No	Supply Chain	Basic	No
21	2022	[45]	Node-RED	Behavior	No	Temperature Tracking of vaccines	Intermediate	No
22	2023	[46]	DCR graphs	Structure	Yes	Casino example	Intermediate	No
23	2023	[32]	Blockly	Structure	Yes	Carbon Offset Hook	Basic	No
24	2023	[47]	BPMN	Behavior	No	Supply Chain Order Process Trade	Advanced	No
25	2023	[48]	BPMN	Behavior	Yes	X-rays exam process	Advanced	No

No.	Year	Reference in bibtext	Visual Approach	Visualization Category	Available Online	Use case	Accessibility Requirements	UX testing
26	2023	[49]	BPMN	Behavior	Yes	Flight delay insurance Voting and Election Voting Result Notification Crowdfunding Supply Chain Buying and Selling	Basic	No
27	2023	[50]	BPMN	Behavior	Yes	Management of birth certificates municipality Visit of pediatric patient hospital Teleconsultation between hospitals Road misconstruction	Advanced	42 users
28	2023	[26]	UML	Behavior	Yes	Certificate issuance	Intermediate	No
29	2024	[51]	Network of Timed Automata	Behavior	No	PhD regulations Business Google TikTok	Advanced	No
30	2024	[52]	BPMN	Evolution	Yes	Office building construction	Intermediate	9 users
31	2024	[53]	Control Flow Graph	Behavior	No	Ponzi scheme detection	Advanced	12 users
32	2024	[34]	Blockly	Structure	No	NFT metadata upload	Basic	17 students
33	2024	[33]	Blockly	Structure	Yes	Learning smart contract's coding	Basic	N/A
34	2024	[54]	BPMN	Behavior	No	Bank loan assessment	Intermediate	No
35	2024	[55]	BPMN	Behavior	No	Certificate management system	Intermediate	No
36	2024	[56]	BPMN	Behavior	Yes	Pizza order Purchase Management system Rental claim Manufactory Supply chain Incident management Blood Analysis Bike Rental Coffee Machine Exam Procedure Hotel Booking	Advanced	No
37	2024	[57]	BPMN	Behavior	No	Multiple-Choice Questions exam management	Intermediate	No
38	2025	[58]	ADOxx	Behavior	Yes	Ontology attestation Decentralized auctions	Intermediate	No

No.	Year	Reference in bibtext	Visual Approach	Visualization Category	Available Online	Use case	Accessibility Requirements	UX testing
39	2025	[59]	BPMN	Behavior	Yes	Land leasing	Advanced	7 students
40	2025	[60]	BPMN	Behavior	No	Voting Services Lending Sales Auctions Purchases Leases	Advanced	No

Table 1: Selected papers in chronological order of publication.

4.2. Model Driven Visualization Approach

BPMN has emerged as the dominant visualization framework for model-driven smart contract development, with researchers adapting its notation for blockchain contexts in various ways. Several approaches focus on choreography and multi-party collaboration. López-Pintado et al. [36] introduced Caterpillar, which uses single-pool BPMN models in which stakeholders appear as lanes, replacing message-based coordination with sequence flows to represent blockchain as a unified coordination mechanism. Similarly, Shen et al. [56] used BPMN choreography diagrams to visualize collaborative processes through message exchanges, with participant bands distinguishing initiators from recipients. Samanipour et al. [59] selected choreography diagrams for their MDAPW3 framework specifically to represent peer-to-peer interactions in decentralized applications. Security and flexibility concerns have been addressed through specialized BPMN extensions. Kopke et al. [50] improved BPMN with blockchain-specific annotations including train symbols for enforceability and chain symbols for on-chain execution. Corradini et al. [48] developed FlexChain, using BPMN choreography diagrams to address the tension between blockchain immutability and business process flexibility. Several researchers have explored transformation methodologies from BPMN to smart contract code. El Abidi et al. [55, 57] developed systematic approaches to translate BPMN elements into Solidity constructs through ranking processes and meta-model integration. Jin et al. [54] employed a two-step transformation process that converts BPMN models into Prolog statements for validation before generating Go language smart contracts. Gao et al. [60] introduced BPMN-LLM, using large language models to transform BPMN diagrams into smart contracts, where swimlanes denote parties and gateway types capture conditional logic. Some approaches combine BPMN with other visualization techniques. Shen et al. [49] integrated customized BPMN modeling with Google Blockly's visual programming environment, distinguishing between participant actions and smart contract operations through specialized swimlanes. Ye et al. [52] developed a three-tier visual framework mapping BPMN tasks to smart contract functions with color-coded indicators reflecting execution states. Alternative state-based approaches include Meng et al.'s [40] EFSMSolid framework, which employs Extended Finite State Machines to represent contractual phases and transitions. Ye and Konig [41] selected YAWL (Yet Another Workflow Language) for its formal semantics while supporting practical control-flow patterns that construction professionals could understand. Industry-specific implementations such as Rosa-Bilbao et al. [45]propose EDALoCo which extends Node-RED to create a modular low-code framework with blockchain-oriented nodes classified by functionality, while Bodorik et al. [47] transformed BPMN through an intermediate Event-Hierarchical State Machine representation to identify patterns for sidechain deployment. Collectively, these BPMN-based approaches demonstrate the notation's versatility in representing smart contract elements – participants, roles, actions, obligations, and constraints – while providing visual abstractions accessible to both technical and non-technical stakeholders.

4.3. Custom Visualization Approach

Several researchers have developed specialized visualization techniques specifically design for smart contracts. Bragagnolo et al. [35] created SmartInspect, which uses a mirror-based reflection system to visualize the contract state through hierarchical tree structures, making complex data interpretable without redeployment. Rather than using standard frameworks, they developed a custom decompilation technique for dynamic representation of contract data.

Node-based interfaces have emerged as a popular approach. Tan et al. [37] introduced LATTE, which represents contract elements as interconnected nodes in action graphs with semantic parsing capabilities. Similarly, Hdhili et al. [43] employed flow-based programming to depict smart contracts as executable graphs of modular boxes showing input-output relationships, demonstrating improved comprehension among non-experts.

Several frameworks target domain experts through specialized visual languages. Bistarelli et al. [44] developed a DSL for supply chain contracts where assets, operations, containers, and roles are represented through standardized graphical components. Curty and Fill [58] proposed SmartCML, using

decorated circles for actions and explicit relations to distinguish between read and write operations on the contract state.

For formal modeling, researchers have explored various notations. Prunell and Schwitter [42] created a Smart Document Editor using color-coded bars and distinctive icons aligned with Answer Set Programming. Eshgie et al. [46] used DCR graphs with styled arrows representing different constraints, while Qin et al. [51] used state transition diagrams to capture contract clauses as networks of states with annotated edges.

Some approaches integrate multiple visualization techniques. Skotnica et al. [38] proposed a meta-model decomposing contract logic into interrelated diagrams, including a Contract Action Model using Google Blockly for executable logic. In subsequent work [39], they introduced DasContract, combining data models, process models, and form models to generate Solidity code. Wen et al. [53] developed a hierarchical visualization system that transforms bytecode into semantic action sequences, using color-coded charts, parallel sequences, and node-link diagrams with specialized iconography.

These custom approaches collectively demonstrate diverse strategies for visualizing smart contracts, emphasizing relationships between parties, representing execution paths, modeling state changes, and abstracting complex programming constructs for improved usability and understanding.

5. Analysis

The analysis of the selected work on smart contract visualization from 2018 to 2025 reveals distinct patterns in the adoption of visualization methodology.

As shown in Figure 1a BPMN represents 35% (14 papers) of the studies, making it the most popular methodology. Blockly and UML follow closely with 17.5% (7 papers) and 20% (8 papers) respectively. The remaining 27.5% (11 papers) utilize various visualization approaches, including DCR graphs, Answer Set Programming, Network of Timed Automata, Flow-based programming, EFSM Model, DSL, Node-RED, JSON/HTML/REST interfaces, and ADOxx.

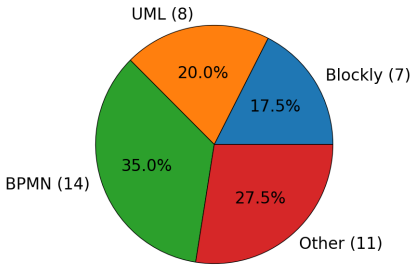
5.1. Temporal Analysis

The temporal distribution of these methodologies, depicted in Figure 1b, reveals a significant evolution in the field. From 2018 to 2020, approaches were dominated by more general-purpose visualization languages such as UML and Blockly, which were favored for their accessibility and familiarity to software developers. This period established the foundational visualization approaches in the emerging field of smart contract visualization. The years 2021-2022 marked a transition period characterized by methodological diversity. Researchers began exploring specialized approaches such as EFSM, DCR graphs, and custom visualizations tailored to the unique requirements of blockchain environments. This diversification signaled a maturing understanding of the specific challenges in visualizing smart contracts. Most notably, from 2023 to 2025, a pronounced shift toward BPMN is observed as the dominant methodology. Of the 14 BPMN-based studies, 11 were published during this recent period. This convergence suggests an emerging consensus around BPMN's suitability for modeling the procedural nature and complex interactions of smart contracts. The standardization around BPMN may also reflect a move toward industry adoption, as BPMN is widely used in business contexts.

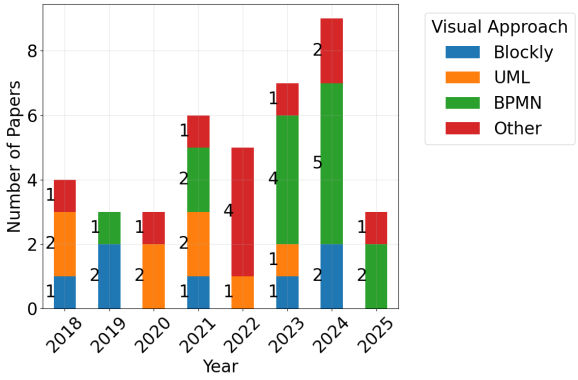
5.2. Visualization Types Analysis

The conducted analysis categorizes the visualizations into three distinct types. Taking as reference the Figure 1c, there is the following distribution:

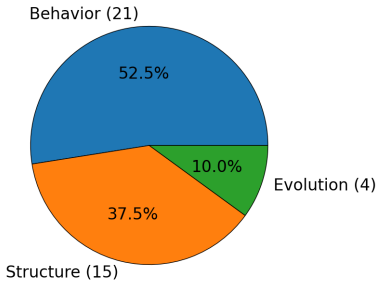
- **Structure** (15 studies, 37.5%): focusing on compositional aspect of smart contract source code;
- **Behavior** (21 studies, 52.5%): emphasizing the dynamic execution and interaction patterns of smart contracts;
- **Evolution** (4 studies, 10%): addressing the temporal dimension and smart contract modifications.



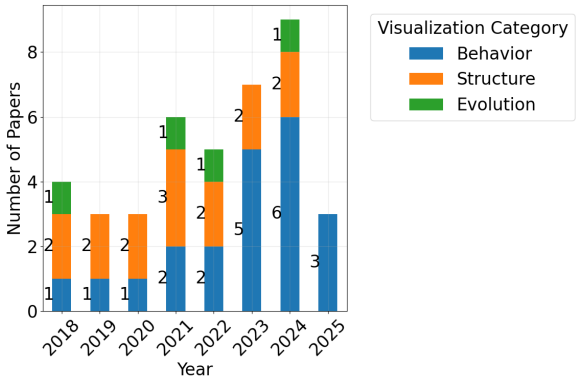
(a) Distribution of Visual Approaches



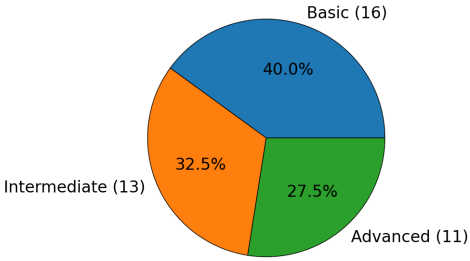
(b) Papers by Year and Visual Approach



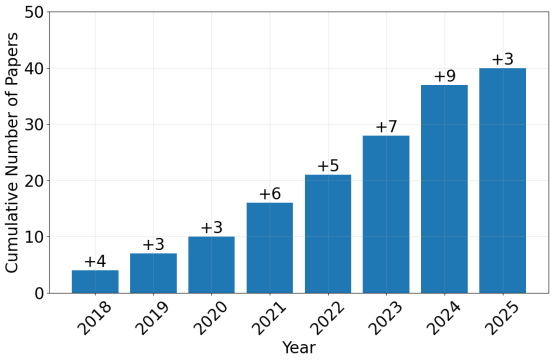
(c) Distribution of Visualization Categories



(d) Papers by Year and Visualization Category



(e) Distribution of Accessibility Requirements



(f) Cumulative Papers by Year

Figure 1: Comprehensive visualization of research trends: (a) illustrates the distribution of visual approaches with BPMN (35%) being most prevalent; (b) displays visualization categories with behavior (52.5%) as the dominant focus; (c) shows accessibility requirements distribution across basic (40%), intermediate (32.5%), and advanced (27.5%) categories; (d) presents the cumulative growth of papers from 2018 to 2025; (e) details the yearly breakdown of papers by visual approach type; and (f) presents the distribution of visualization categories over time, showing an increasing focus on behavior visualization in recent years.

The distribution in Figure 1d shows a progressive shift from structural representations toward behavioral ones. Early research (2018-2021) predominantly focused on structural visualization (11 out of 15 structure-focused papers are published in this period), which addresses the fundamental question of how smart contracts are composed. As the field matured, particularly from 2022 onward, behavioral visualizations gained prominence (16 out of 21 behavior-focused papers published in this period),

reflecting an increased interest in how contracts execute and interact with users and other contracts. This transition aligns with the natural progression of the field: researchers first needed to establish how to represent what smart contracts are before moving on to visualizing how they behave. The limited number of evolution-focused visualizations (10%) suggests an opportunity for future research, particularly as contract upgradability and adaptation become more critical in production environments.

5.3. Accessibility Requirements Analysis

The distribution of accessibility requirements in the studies depicted in Figure 1e reveals a balanced approach to user expertise:

- **Basic**: 16 studies (40%);
- **Intermediate**: 13 studies (32.5%);
- **Advanced**: 11 studies (27.5%).

This relatively uniform distribution suggests that researchers are considering various user profiles, from novices to experienced developers. In particular, block-based visualizations predominantly target basic accessibility levels, making them suitable for educational purposes and newcomers to smart contract development. In contrast, BPMN-based approaches tend toward intermediate and advanced requirements, reflecting their use in more complex scenarios. The analysis revealed a concerning gap in user experience validation. As Table 1 highlights, only 8 out of 40 studies (20%) incorporated formal UX testing, most of which involve relatively small sample sizes (6-17 participants). Notable exceptions include studies by Hdhili et al. [43] and Kopke et al. [50], with 58 and 42 participants, respectively. This limited focus on user experience suggests that smart contract visualization remains primarily in the technical development phase, with insufficient attention to end-user needs and usability. However, we observe a positive trend in recent years (2023-2025), where 6 of the 8 studies with UX testing were published, indicating a growing recognition of user-centered design principles.

5.4. Use Case Analysis

The use cases that span the 40 studies demonstrate remarkable diversity, as reported in Table 1, including:

- **Financial Services**: voting systems, auctions, lending platforms, insurance claims;
- **Supply Chain Management**: product tracking, trade verification, manufacturing processes;
- **Real Estate**: property transactions, mortgage processing, leasing agreements;
- **Healthcare**: patient records, hospital procedures, tele-consultation;
- **Education**: certificate issuance, exam management;
- **General Business Processes**: payment systems, purchasing workflows.

This broad application spectrum indicates that smart contract visualization is being explored across multiple domains rather than being confined to specific industries. The diversity of use cases underscores the versatile applicability of smart contracts and the universal need for effective visualization regardless of the application domain.

5.5. General Trends

In general, looking the Figure 1f showing cumulative papers by year, a steady growth in research publications proposing visual methods for smart contract source code representation can be observed. Starting with just 4 papers in 2018, the field has shown consistent annual growth, with particularly strong publication increases in 2021 (+6 papers), 2023 (+7 papers), and 2024 (+9 papers). By 2025, the cumulative total reaches 40 publications, representing a ten-fold increase from the initial baseline. This growth pattern suggests increasing research interest in visual representations of smart contracts, likely driven by the expanding adoption of blockchain technologies and the need for more accessible ways to understand and verify complex smart contract code. The acceleration in publications from 2021 onward

may reflect the maturation of the field as researchers build upon earlier visualization approaches and address more sophisticated representation challenges. The consistent year-over-year increases, rather than sporadic jumps, indicate sustained interest rather than temporary research trends, suggesting that this is becoming an established research domain within the broader fields of software visualization and blockchain technology.

6. Conclusions

The analysis carried out aims to map the visualization formalisms and techniques presented so far by the academia to represent smart contracts, self-executing pieces of code that run on the blockchain. After a search phase on the Scopus database, 40 papers have been taken into account to perform the analysis, revealing a clear evolutionary trajectory in smart contract visualization from 2018 to 2025. The field has progressed from structure-focused to behavior-focused visualizations, with an emerging convergence around BPMN as a preferred methodology. Despite this maturation, significant opportunities remain for improving user experience evaluation and developing visualizations that address contract evolution. These findings suggest that while the technical foundations of smart contract visualization are becoming well-established, additional research is needed to bridge the gap between technical capability and practical usability. The balanced distribution across accessibility levels indicates awareness of diverse user needs, but the limited UX testing highlights the need for greater emphasis on user-centered design approaches as these visualization methodologies transition from academic exploration to practical industry adoption.

Acknowledgments

The work discussed in this paper has been supported by the B4A - Blockchain for All project (https://www.blockchain4all.it/), ref. no. 20225MN5K3. This project has been funded with support from the Ministry of Education, University and Research. This document reflects the views only of the authors, and the Ministry of Education, University and Research cannot be held responsible for any use which may be made of the information contained therein.

Declaration on Generative AI

During the preparation of this work, the author(s) used Claude-3.7-Sonnet and Deepl to perform grammar and spelling checks. After using these tools, the authors reviewed and edited the content as needed and assumed full responsibility for the content of the publication.

References

[1] Z. A. Khan, A. S. Namin, A survey of vulnerability detection techniques by smart contract tools, IEEE Access 12 (2024) 70870–70910. doi:10.1109/ACCESS.2024.3401623.

[2] N. He, L. Wu, H. Wang, Y. Guo, X. Jiang, Characterizing Code Clones in the Ethereum Smart Contract Ecosystem, in: J. Bonneau, N. Heninger (Eds.), Financial Cryptography and Data Security, Springer International Publishing, Cham, 2020, pp. 654–675.

[3] S. Diehl, Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software, Springer Berlin Heidelberg, 2007. URL: https://books.google.it/books?id=8OADPxGJOa8C.

[4] N. Chotisarn, L. Merino, X. Zheng, S. Lonapalawong, T. Zhang, M. Xu, W. Chen, A systematic literature review of modern software visualization, Journal of Visualization 23 (2020) 539–558. URL: https://doi.org/10.1007/s12650-020-00647-w. doi:10.1007/s12650-020-00647-w.

[5] D. Hayatpur, D. Wigdor, H. Xia, Crosscode: Multi-level visualization of program execution, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, CHI '23,

Association for Computing Machinery, New York, NY, USA, 2023. URL: https://doi.org/10.1145/3544548.3581390. doi:10.1145/3544548.3581390.

[6] K. E. Isaacs, T. Gamblin, Preserving command line workflow for a package management system using ascii dag visualization, IEEE Transactions on Visualization and Computer Graphics 25 (2019) 2804–2820. doi:10.1109/TVCG.2018.2859974.

[7] S. Devkota, K. E. Isaacs, CFGExplorer: Designing a Visual Control Flow Analytics System around Basic Program Analysis Operations, Computer Graphics Forum 37 (2018) 453–464. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.13433. doi:10.1111/cgf.13433, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13433.

[8] Y. Yoon, B. A. Myers, S. Koo, Visualization of fine-grained code change history, in: 2013 IEEE Symposium on Visual Languages and Human Centric Computing, 2013, pp. 119–126. doi:10.1109/VLHCC.2013.6645254.

[9] F. Zhou, Y. Fan, S. Lv, L. Jiang, Z. Chen, J. Yuan, F. Han, H. Jiang, G. Bai, Y. Zhao, Fctree: Visualization of function calls in execution, Inf. Softw. Technol. 175 (2024). URL: https://doi.org/10.1016/j.infsof.2024.107545. doi:10.1016/j.infsof.2024.107545.

[10] K. E. Isaacs, P.-T. Bremer, I. Jusufi, T. Gamblin, A. Bhatele, M. Schulz, B. Hamann, Combing the communication hairball: Visualizing parallel execution traces using logical time, IEEE Transactions on Visualization and Computer Graphics 20 (2014) 2349–2358. doi:10.1109/TVCG.2014.2346456.

[11] K. Xu, Y. Wang, L. Yang, Y. Wang, B. Qiao, S. Qin, Y. Xu, H. Zhang, H. Qu, Clouddet: Interactive visual analysis of anomalous performances in cloud computing systems, IEEE Transactions on Visualization and Computer Graphics 26 (2020) 1107–1117. doi:10.1109/TVCG.2019.2934613.

[12] Y. Sun, Y. Zhang, A. Mosallaei, M. D. Shah, C. Dunne, D. Kaeli, Daisen: A Framework for Visualizing Detailed GPU Execution, Computer Graphics Forum 40 (2021) 239–250. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14303. doi:10.1111/cgf.14303, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14303.

[13] Q. Yu, K. Yu, B. Li, Effects of Block-Based Visual Programming on K-12 Students' Learning Outcomes, Journal of Educational Computing Research 63 (2025) 64–98. URL: https://doi.org/10.1177/07356331241293163. doi:10.1177/07356331241293163, publisher: SAGE Publications Inc.

[14] D. Pereira, F. Barbosa, C. Morgado, NextBlocks: An Interactive Block Programming Platform, in: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, pp. 590–596. URL: https://dl.acm.org/doi/10.1145/3649217.3653609. doi:10.1145/3649217.3653609.

[15] M. Seraj, M. Verano Merino, E. Rahimi, L. Ochoa Venegas, Programming smart objects: How young learners' programming skills, attitudes, and perception are influenced, in: Proceedings of the 2024 ACM SIGPLAN International Symposium on SPLASH-E, SPLASH-E '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 45–55. URL: https://doi.org/10.1145/3689493.3689982. doi:10.1145/3689493.3689982.

[16] S. Thorgeirsson, T. B. Weidmann, K.-H. Weidmann, Z. Su, Comparing cognitive load among undergraduate students programming in python and the visual language algot, in: Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 1328–1334. URL: https://doi.org/10.1145/3626252.3630808. doi:10.1145/3626252.3630808.

[17] S. Thorgeirsson, C. Zhang, T. B. Weidmann, K.-H. Weidmann, Z. Su, An Electroencephalography Study on Cognitive Load in Visual and Textual Programming, in: Proceedings of the 2024 ACM Conference on International Computing Education Research - Volume 1, ACM, Melbourne VIC Australia, 2024, pp. 280–292. URL: https://dl.acm.org/doi/10.1145/3632620.3671124. doi:10.1145/3632620.3671124.

[18] S. Thorgeirsson, O. Graf, Explaining Algorithms with the Visual Programming Language Algot, in: Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 2, ITiCSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, pp. 783–784. URL: https://dl.acm.org/doi/10.1145/3649405.3659520. doi:10.1145/3649405.3659520.

[19] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, A. Hobor, Making smart contracts smarter, in: Proceedings

of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 254–269. URL: https://doi.org/10.1145/2976749.2978309. doi:10.1145/2976749.2978309.

[20] M. L. L. Vieira, P. Vilain, Representation of smart contracts as state diagrams, in: 2021 IEEE/ACS 18th International Conference on Computer Systems and Applications (AICCSA), 2021, pp. 1–8. doi:10.1109/AICCSA53542.2021.9686862.

[21] S. Curty, F. Härer, H. Fill, Design of blockchain-based applications using model-driven engineering and low-code/no-code platforms: a structured literature review, Softw. Syst. Model. 22 (2023) 1857–1895. URL: https://doi.org/10.1007/s10270-023-01109-1. doi:10.1007/S10270-023-01109-1.

[22] M. Marchesi, L. Marchesi, R. Tonelli, An agile software engineering method to design blockchain applications, in: Proceedings of the 14th Central and Eastern European Software Engineering Conference Russia, CEE-SECR '18, Association for Computing Machinery, New York, NY, USA, 2018. URL: https://doi.org/10.1145/3290621.3290627. doi:10.1145/3290621.3290627.

[23] P. Garamvölgyi, I. Kocsis, B. Gehl, A. Klenik, Towards model-driven engineering of smart contracts for cyber-physical systems, in: 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), 2018, pp. 134–139. doi:10.1109/DSN-W.2018.00052.

[24] G. A. Pierro, Smart-graph: Graphical representations for smart contract on the ethereum blockchain, in: 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2021, pp. 708–714. doi:10.1109/SANER50967.2021.00090.

[25] R. Heckel, Z. Erum, N. Rahmi, A. Pul, Visual smart contracts for daml, in: N. Behr, D. Strüber (Eds.), Graph Transformation, Springer International Publishing, Cham, 2022, pp. 137–154.

[26] M. Jurgelaitis, L. Čeponienė, K. Butkus, R. Butkienė, V. Drungilas, Mda-based approach for blockchain smart contract development, Applied Sciences 13 (2023). URL: https://www.mdpi.com/2076-3417/13/1/487. doi:10.3390/app13010487.

[27] M. Ghaffari Saadat, R. Heckel, F. Orejas, Unfolding symbolic attributed graph grammars, in: F. Gadducci, T. Kehrer (Eds.), Graph Transformation, Springer International Publishing, Cham, 2020, pp. 75–90.

[28] T. Weingaertner, R. Rao, J. Ettlin, P. Suter, P. Dublanc, Smart Contracts Using Blockly: Representing a Purchase Agreement Using a Graphical Programming Language, in: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), 2018, pp. 55–64. URL: https://ieeexplore.ieee.org/document/8525393. doi:10.1109/CVCBT.2018.00012.

[29] L. Guida, F. Daniel, Supporting reuse of smart contracts through service orientation and assisted development, in: 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON), 2019, pp. 59–68. doi:10.1109/DAPPCON.2019.00017.

[30] D. Mao, F. Wang, Y. Wang, Z. Hao, Visual and User-Defined Smart Contract Designing System Based on Automatic Coding, IEEE Access 7 (2019) 73131–73143. URL: https://ieeexplore.ieee.org/document/8730359. doi:10.1109/ACCESS.2019.2920776.

[31] M. M. Merlec, Y. K. Lee, H. P. In, SmartBuilder: A Block-based Visual Programming Framework for Smart Contract Development, in: 2021 IEEE International Conference on Blockchain (Blockchain), 2021, pp. 90–94. URL: https://ieeexplore-ieee-org.ezproxy.biblio.polito.it/document/9680565. doi:10.1109/Blockchain53845.2021.00023.

[32] L. A. Trestioreanu, W. M. Shbair, F. S. de Cristo, R. State, Blockly2Hooks: Smart Contracts for Everyone with the XRP Ledger and Google Blockly, in: 2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS), 2023, pp. 145–150. URL: https://ieeexplore-ieee-org.ezproxy.biblio.polito.it/document/10237017. doi:10.1109/DAPPS57946.2023.00027, iSSN: 2835-3498.

[33] C. Gomez, J. Vara, F. Pérez-Blanco, D. Granada, A block-based web ide to ease the smart contract programming learning curve, IEEE Revista Iberoamericana de Tecnologias del Aprendizaje PP (2024) 1–1. doi:10.1109/RITA.2024.3487475.

[34] Y.-C. Tsai, J.-Y. Huang, D.-R. Chiou, Empowering young learners to explore blockchain with user-friendly tools: a method using Google Blockly and NFTs, IET Blockchain 4 (2024) 324–334. URL:

https://onlinelibrary.wiley.com/doi/abs/10.1049/blc2.12055. doi:10.1049/blc2.12055, _eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/blc2.12055.

[35] S. Bragagnolo, H. Rocha, M. Denker, S. Ducasse, Smartinspect: solidity smart contract inspector, in: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2018, pp. 9–18. doi:10.1109/IWBOSE.2018.8327566.

[36] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, A. Ponomarev, Caterpillar: A business process execution engine on the Ethereum blockchain, Software: Practice and Experience 49 (2019) 1162–1193. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2702. doi:10.1002/spe.2702, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2702.

[37] S. Tan, S. S Bhowmick, H. E. Chua, X. Xiao, LATTE: Visual Construction of Smart Contracts, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, SIGMOD '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 2713–2716. URL: https://dl.acm.org/doi/10.1145/3318464.3384687. doi:10.1145/3318464.3384687.

[38] M. Skotnica, R. Pergl, Das contract - a visual domain specific language for modeling blockchain smart contracts, in: D. Aveiro, G. Guizzardi, J. Borbinha (Eds.), Advances in Enterprise Engineering XIII, Springer International Publishing, Cham, 2020, pp. 149–166.

[39] M. Skotnica, J. Klicpera, R. Pergl, Towards model-driven smart contract systems-code generation and improving expressivity of smart contract modeling, Proc. EEWC 20 (2020).

[40] J. Meng, Z. Li, R. Zhao, Y. Shang, An Automated Modeling Method and Visualization Implementation of Smart Contracts, in: H.-N. Dai, X. Liu, D. X. Luo, J. Xiao, X. Chen (Eds.), Blockchain and Trustworthy Systems, Springer, Singapore, 2021, pp. 399–406. doi:10.1007/978-981-16-7993-3_30.

[41] X. Ye, M. König, From the graphical representation to the smart contract language: a use case in the construction industry, in: C. Feng, T. Linner, I. Brilakis, D. Castro, P.-H. Chen, Y. Cho, J. Du, S. Ergan, B. Garcia de Soto, J. Gaparík, F. Habbal, A. Hammad, K. Iturralde, T. Bock, S. Kwon, Z. Lafhaj, N. Li, C.-J. Liang, B. Mantha, M. S. Ng, D. Hall, M. Pan, W. Pan, F. Rahimian, B. Raphael, A. Sattineni, C. Schlette, I. Shabtai, X. Shen, P. Tang, J. Teizer, Y. Turkan, E. Valero, Z. Zhu (Eds.), Proceedings of the 38th International Symposium on Automation and Robotics in Construction (ISARC), International Association for Automation and Robotics in Construction (IAARC), Dubai, UAE, 2021, pp. 272–279. doi:10.22260/ISARC2021/0039.

[42] K. Purnell, R. Schwitter, User-defined smart contracts using answer set programming, in: G. Long, X. Yu, S. Wang (Eds.), AI 2021: Advances in Artificial Intelligence, Springer International Publishing, Cham, 2022, pp. 291–303.

[43] F. Hdhili, R. Gouiaa, M. Jansen, Implementation and evaluation of a visual programming language in the context of blockchain, in: J. Prieto, A. Partida, P. Leitão, A. Pinto (Eds.), Blockchain and Applications, Springer International Publishing, Cham, 2022, pp. 74–82.

[44] S. Bistarelli, F. Faloci, P. Mori, Towards a graphical dsl for tracing supply chains on blockchain, in: R. Chaves, D. B. Heras, A. Ilic, D. Unat, R. M. Badia, A. Bracciali, P. Diehl, A. Dubey, O. Sangyoon, S. L. Scott, L. Ricci (Eds.), Euro-Par 2021: Parallel Processing Workshops, Springer International Publishing, Cham, 2022, pp. 219–229.

[45] J. Rosa-Bilbao, J. Boubeta-Puig, A. Rutle, Edaloco: Enhancing the accessibility of blockchains through a low-code approach to the development of event-driven applications for smart contract management, Computer Standards Interfaces 84 (2022) 103676. doi:10.1016/j.csi.2022.103676.

[46] M. Eshghie, W. Ahrendt, C. Artho, T. T. Hildebrandt, G. Schneider, Capturing smart contract design with dcr graphs, in: C. Ferreira, T. A. C. Willemse (Eds.), Software Engineering and Formal Methods, Springer Nature Switzerland, Cham, 2023, pp. 106–125.

[47] P. Bodorik, C. G. Liu, D. Jutla, TABS: Transforming automatically BPMN models into blockchain smart contracts, Blockchain: Research and Applications 4 (2023) 100115. URL: https://www.sciencedirect.com/science/article/pii/S2096720922000562. doi:10.1016/j.bcra.2022.100115.

[48] F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, F. Tiezzi, A flexible approach to multi-party business process execution on blockchain, Future Generation Computer Systems 147

(2023) 219–234.

[49] X. Shen, W. Li, H. Xu, X. Wang, Z. Wang, A reuse-oriented visual smart contract code generator for efficient development of complex multi-party interaction scenarios, Applied Sciences 13 (2023) 8094.

[50] J. Köpke, G. Meroni, M. Salnitri, Designing secure business processes for blockchains with secbpmn2bc, Future Generation Computer Systems 141 (2023) 382–398.

[51] P. Qin, Q. Hu, M. Cui, Towards machine-readable semantic-based e-business contract representations using network of timed automata (nta), Future Generation Computer Systems 158 (2024) 457–471. URL: https://www.sciencedirect.com/science/article/pii/S0167739X24001699. doi:https://doi.org/10.1016/j.future.2024.04.040.

[52] X. Ye, N. Zeng, X. Tao, D. Han, M. König, Smart contract generation and visualization for construction business process collaboration and automation: Upgraded workflow engine, Journal of Computing in Civil Engineering 38 (2024) 04024030. URL: https://ascelibrary.org/doi/abs/10.1061/JCCEE5.CPENG-5938. doi:10.1061/JCCEE5.CPENG-5938. arXiv:https://ascelibrary.org/doi/pdf/10.1061/JCCEE5.CPENG-5938.

[53] X. Wen, T. D. Nguyen, S. Ruan, Q. Shen, J. Sun, F. Zhu, Y. Wang, Ponzilens+: Visualizing bytecode actions for smart ponzi scheme identification, IEEE Transactions on Visualization and Computer Graphics (2024) 1–14. doi:10.1109/TVCG.2024.3516379.

[54] J. Jin, L. Yan, Y. Zou, J. Li, Z. Yu, Research on smart contract verification and generation method based on bpmn, Mathematics 12 (2024) 2158.

[55] A. El Abidi, H. Elghazi, Converting bpmn diagrams into solidity code: A comprehensive new approach and algorithm, in: 2024 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD), IEEE, 2024, pp. 1–6.

[56] X. Shen, Z. Wang, J. Luo, H. Ruan, H. Xu, M. Liu, Ibc: An integrated framework combining blockchain with bpmn choreography to enhance multi-party collaboration, in: 2024 IEEE International Conference on Web Services (ICWS), IEEE, 2024, pp. 457–467.

[57] A. El Abidi, H. El Ghazi, S. Assar, An integration model for the mapping of bpmn to smart contract: Mcq case study, IEEE Engineering Management Review (2024).

[58] S. Curty, H.-G. Fill, Smartcml: A visual modeling language to enhance the comprehensibility of smart contract implementations, in: E. Paja, J. Zdravkovic, E. Kavakli, J. Stirna (Eds.), The Practice of Enterprise Modeling, Springer Nature Switzerland, Cham, 2025, pp. 87–104.

[59] A. Samanipour, O. Bushehrian, G. Robles, Mdapw3: Mda-based development of blockchain-enabled decentralized applications, Science of Computer Programming 239 (2025) 103185.

[60] S. Gao, W. Liu, J. Zhu, X. Dong, J. Dong, Bpmn-llm: Transforming bpmn models into smart contracts using large language models, IEEE Software (2025).