Blockly2Hooks: Smart Contracts for Everyone with the XRP Ledger and Google Blockly

Lucian Trestioreanu*, Wazen M. Shbair*, Flaviene Scheidt de Cristo*, and Radu State*
* University of Luxembourg, SnT, 29, Avenue J.F Kennedy, L-1855 Luxembourg
Email:{lucian.trestioreanu, wazen.shbair, radu.state}@uni.lu

***Abstract*—Recent technologies such as inter-ledger payments, non-fungible tokens, and smart contracts are all fruited from the ongoing development of Distributed Ledger Technologies. The foreseen trend is that they will play an increasingly visible role in daily life, which will have to be backed by appropriate operational resources. For example, due to increasing demand, smart contracts could soon face a shortage of knowledgeable users and tools to handle them in practice. Widespread smart contract adoption is currently limited by security, usability and costs aspects. Because of a steep learning curve, the handling of smart contracts is currently performed by specialised developers mainly, and most of the research effort is focusing on smart contract security, while other aspects like usability being somewhat neglected. Specific tools would lower the entry barrier, enabling interested non-experts to create smart contracts.**

In this paper we designed and developed Blockly2Hooks, a solution towards filling this gap even in challenging scenarios such as when the smart contracts are written in an advanced language like C. With the XRP Ledger as a concrete working case, Blockly2Hooks helps interested non-experts from the community to learn smart contracts easily and adopt the technology, through leveraging well-proven teaching methodologies like Visual Programming Languages, and more specifically, the Blockly Visual Programming library from Google.

The platform was developed and tested and the results are promising to make learning smart contract development smoother.

***Index Terms*—DLT, XRP, smart contracts, visual programming**

I. Introduction

The recent advances and the growing adoption of the *Distributed Ledger Technology (DLT)* show that DLT is here to stay, with the technology becoming each day more, a part of the daily life. Inter-ledger payments [1] made their way into finances [2], non-fungible tokens (NFT) [3]–[5] are already used in industries like gaming, logistics and more; smart contracts [6], [7] are being successfully deployed to tackle finance, healthcare, gaming, insurance or legal use cases [8]. The term *"smart contracts"* was coined in 1994 by Nick Szabo who defined them as software programs replicating real-life contracts and executing their terms while minimizing exceptions and the need for trusted intermediaries, e.g. notaries [9], [10]. Smart contracts [11] are small pieces of executable code, oftentimes written in advanced programming languages. They reside and execute on DLTs like Ethereum (ETH) [12], [13] or the XRP Ledger (XRPL) [14], [15]. Because they exist and run on DLT, they are classified as Decentralized Applications (dApps). Ideally, smart contracts should be autonomous, immutable and public; their execution should be transparent and non-reversible. On ETH, the smart contract's logic is encoded with Solidity, compiled to bytecode, then executed on the Ethereum Virtual Machine [16]. Solidity is a high level, object-oriented programming language designed specifically to encode smart contracts on the Ethereum blockchain[1]. Initially inspired from JavaScript, it was enriched with elements from C++ and Python. On XRPL, smart contracts are written in C, compiled to *wasm* (a binary format for a stack-based virtual machine)[2], then deployed and executed on XRPL. However, the learning curve for smart contract programming is not smooth. Currently, the smart contracts are mostly written, debugged and deployed by specialised developers with advanced technical background. Besides advanced knowledge in the specialized field, another important reason is that smart contracts are written in programming languages like C that can make the process overwhelmingly difficult for non-expert users. Also, the design of the C language is identified as impacting the bug rates, program security and complexity [17].

Smart contracts should be made friendlier and more accessible to a larger audience like teenagers, non-technical people, knowledge seekers and the community at large: in a scenario when smart contracts are deployed on a large scale in industry, it will be desirable that users and stakeholders can directly program the conditions of the contract themselves. Bringing smart contracts closer to the business will unlock the full potential of smart contracts to creating value for the society. It is expected that if adopted on a large scale, the financial impact of smart contracts would be substantial. However, widespread smart contract adoption is currently hampered by costs, usability and security [18]. The specialised programmers needed for smart contract development increase the cost to deploy a smart contract. To enable widespread adoption, costs can be decreased by increasing smart contract usability, currently impacted by the advanced programming languages used in development that translate into an entry barrier for the general public. Moreover, when smart contracts are developed by pure programmers, situations can arise when the programmer does not fully understand the needs of the stakeholder, that should be embedded into the contract [19]. The stakeholder needs to test the implementation and provide feedback to the programmer making the process iterative and

[1]https://docs.soliditylang.org/en/v0.8.18/, valid in February 2023
[2]https://webassembly.org/, valid in February 2023

prone to implementation flaws which increases the costs even further. Currently, there is a general trend towards no-code or low-code tools [20], [21] for software development to benefit ubiquitous, daily-business use cases [22]. The blockchains whose smart contracts will be more usable and cost-friendly will see their adoption and user-base grow.

Of the possible solutions, we focused on the well-proven *Visual Programming Languages (VPL)* methodology which is successfully used in schools to teach programming. VPLs, or *block coding*, have been defined as programming languages that enable users to build computer programs through interacting with, and expressing the program through, graphical elements of the language rather than textual [23]. Oftentimes such environments are based on boxes, lines and arrows which interconnect the boxes and represent their relations and interactions [24], [25]. VPLs aim to make programming more accessible to novices through *syntax, semantics and pragmatics* [26]. *Syntax* offers icons, blocks, arrows, forms and diagrams to enable beginners to *easily* create *well-formed* computer programs. *Semantics* are helper methods aiming to convey to the user the meaning and correct usage of the graphical primitives of the VPL. *Pragmatics* are possibilities offered to the user to test and understand the behavior of pieces of program they create, in different specific situations. VPLs can be used for a variety of domains like multimedia, education, gaming or automation, to name a few.

Visual editors for smart contracts can democratize the building, deployment and management of smart contracts and improve their reliability and security [27], because, like any other computer program, smart contracts can have sometimes bugs that can lead to potentially significant financial losses [28].

What we want to achieve is illustrated in Figure 1: the complex task to write smart contracts in the C code (Figure 1a) and then deploy, test and use them, should be made accessible to larger audiences through leveraging the advantages of visual programming (Figure 1b).

In this paper we propose, implement and discuss Blockly2Hooks, a solution for teaching non-technical audiences how to easily program, test and deploy some of the most challenging smart contracts like those written in the C language. To achieve this we use Visual Programming, a well-proven teaching methodology, and take the XRP Ledger smart contracts as a concrete case. As Google Blockly offers native support for the translation of the visual blocks to simpler languages like JavaScript, Python, PHP, Lua, Dart, and the possibility to add other languages, other blockchains can be accommodated on this design, too.

The paper is organised as follows: Section II introduces the technologies involved. The Blockly2Hooks solution is presented in Section III, while the current state of the art is discussed in Section IV. Finally, we draw our conclusions and present the avenues for future work in Section V.

II. Background

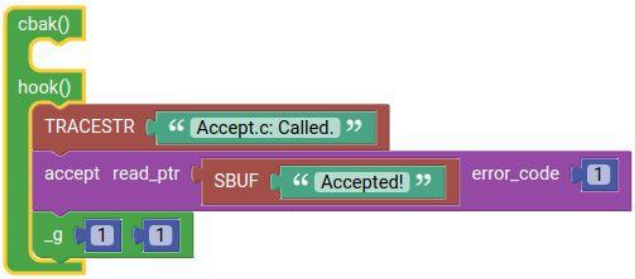
Here we are describing the most important technologies involved in Blockly2Hooks, like the *XRP Ledger*, the *Hooks*

```
//Executed when an emitted transaction is successfully
accepted into a ledger
int64_t cbak(uint32_t reserved) {
 return 0;
}

//Executed whenever a transaction comes into or leaves
from the account the Hook is set on
int64_t hook(uint32_t reserved) {
TRACESTR("Accept.c: Called.");
accept(SBUF("Accepted!"),1);
_g(1,1);

 return 0;
}
```

(a) *XRPL* smart contracts, as written by programmers.



(b) XRPL smart contracts, as developed by non-technical users.

Fig. 1: The design goal of Blockly2Hooks.

smart contracts on XRPL, and the *Blockly* library for visual programming offered by Google.

XRPL is characterised as an open-source, permissionless, and decentralized blockchain which is appreciated for its transaction (tx) throughput (1500 tx/s) [29], speed (transactions settle in 3-5s) [30], low fees, and low energy consumption, all thanks to the consensus protocol involved: the ledger building process consists of a Byzantine Fault Tolerant *"Consensus"* [31] and a *"Validation"* stage, where a majority of the participating nodes have to agree on the next version of the ledger. This is not a computationally-intensive process and it is designed to provide the above mentioned advantages.

XRPL is focused on cross-border payments and has support for NFTs and for smart contracts which on XRPL are called *Hooks* [32].

The **Hooks** have been developed specially for the XRPL. They are small, efficient pieces of code compiled to web assembly (wasm) modules. Hooks can be written in any language (compilable to wasm) then they are uploaded to XRPL [33]. They are deployed and work on Layer 1, meaning directly on the XRPL, and their function is to modify the behavior and the flow of the transactions. The logic they deploy can be executed before or after the transactions. Hooks are deliberately made not to be Turing-Complete, which is undesirable at layer 1

because this would make it impossible to determine when the program would end. And without predictable maximum execution times, the XRPL might never advance to the next ledger. Hooks can store simple data objects like for example lists: *"for all incoming payments, check if the sending account is in a black**list** (e.g. kept by another hook), and if yes: reject it"*. Other possible hook examples are: *"deny transfers less than 20 XRP"*, or *"for outgoing transfers, send xy% to a predefined account"*. Typically, Hooks are written in C. While C is a very efficient programming language, it is not easy to learn and use for non-developers. Even more, because Hooks are deliberately not Turing-Complete programs, they use a modified C language which makes it even more complicated for non-specialised audience.

An example of an XRPL Hook, called *"Carbon Offset example"* [34], is illustrated in Figure 2: The hook in this example is installed on the *Sender* account (Bob's account). This hook is triggered by outgoing transactions from Bob's account: when Bob sends some funds to Alice, the Hook will trigger a new transaction that will send 1% of the outgoing amount to a *"Carbon Offset"* account. It is possible to install such hooks on other sending user accounts too, such that over time the "Carbon Offset" account would raise an amount that can be used to mitigate the effects of carbon emissions on *Global Warming* through for example sponsoring reforestation.

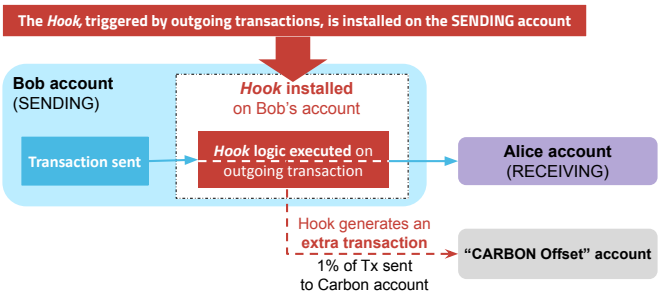


Fig. 2: Example Hook on the XRP Ledger: "Carbon offset".

VPLs. Scratch [35] is a visual programming language developed at the MIT Media Lab. Scratch is used by thousands of kids worldwide to learn how to program and allows them to share their creations with one another over the Internet. The approach, its advantages and what made it so successful is thoroughly explained by the authors in [36]. Nevertheless, Scratch is not easily customizable, and it only translates to Java Script. Another VPL example is Droplet [37] but it is not mature enough, nor widely used. Taleblazer [38] is a platform for developing Augmented Reality games using visual programming but it is specialized on gaming. As such, we found Google Blockly to be more suitable for the purpose:

Blockly [39] is an open-source framework from Google featuring visual block-based icons and a drag-drop programming environment. Non-expert users can leverage the visual programming language approach provided by Blockly to build applications for education, gaming, robotics, or IoT. We chose Blocky because it can transform visual programs into many different textual codes, e.g., JavaScript, Php, Python, Dart, and Lua, and because it is much more versatile and customizable.

III. Methodology and Result

This section describes Blockly2Hooks, the proposed solution for bringing smart contracts closer to large, non-expert audiences. As stated, the goal is to enable everyone interested, to develop and deploy smart contracts even in challenging cases such as when the contracts are written in complex programming languages like C - as currently encountered for example on the XRP Ledger. We do this through leveraging the *Visual Programming Language* (VPL) approach, more specifically by using the visual programming libraries offered by Google that are called *Blockly*.

To achieve this, we first study, map and implement the functions that we could identify in the XRPL Hooks smart contracts as visual *blocks*, using Blockly. This enables the use of the "Drag & Drop" visual programming approach for smart contract development. Next, we implement the compiling of the generated code to Web Assembly (WASM). Finally, through the push of a button after compilation, we enable users to easily sign and deploy to the blockchain, i.e. the XRP Ledger, the smart contract (hook) that they developed using our Blockly-based VPL.

The proposed system architecture is represented in Figure 3, and it comprises four major modules:

- The *Frontend*
- The *Remote Server* for code compilation to web assembly
- The *Backend*
- The *XRPL Hooks Testnet*

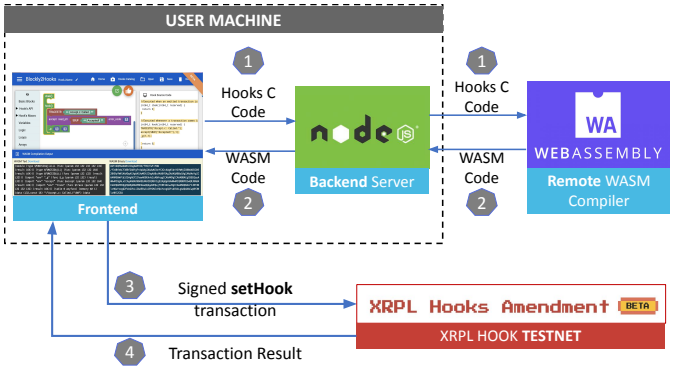


Fig. 3: Blockly2Hooks architecture.

These modules are described below:

Frontend. The user builds smart contracts through interacting with a web interface in their browser. The Frontend features a visual programming environment where users without advanced programming skills can *drag and drop* blocks and fill in basic data (e.g. amounts) needed to build their desired *Hook Smart Contract*. The result is translated to Hooks' C code and can be seen on the same screen on the Frontend, as illustrated in Figure 4.

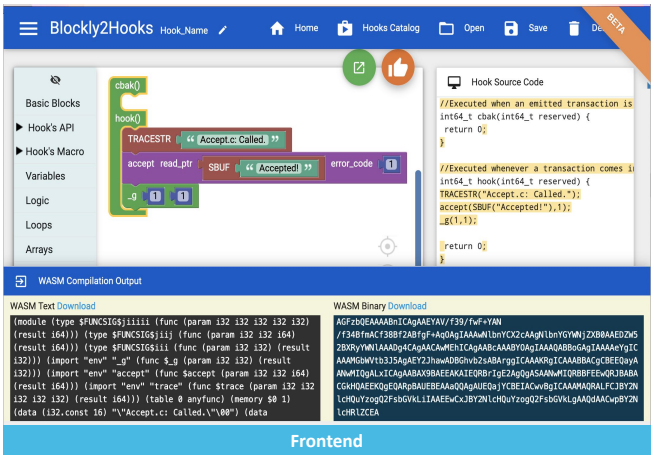


Fig. 4: Blockly2Hooks Frontend.

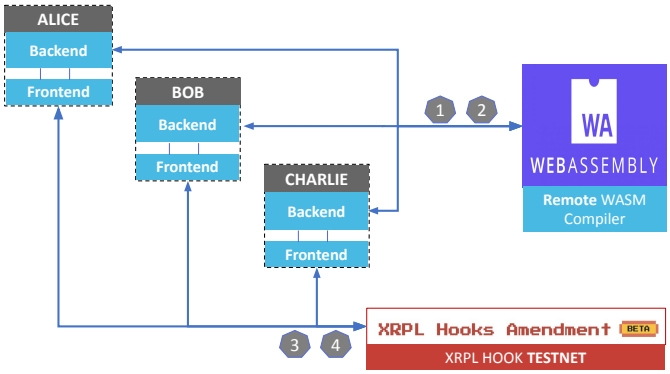


Fig. 5: Blockly2Hooks multi-user architecture.

Remote wasm compiler server. This is a remote server which receives the hooks that have been visually built by the user in the form of C code, and then compiles them to wasm, returning either the same hooks but in the wasm format, or the error(s) encountered during compilation. We chose to externalise the wasm compiler to keep the user experience light and smooth, because the wasm compiler involves the management of complex libraries required to compile the hooks from the special XRPL's C language format to the wasm format. As such, the user needs to fetch and install only the light code associated to the *backend* and the *frontend*.

XRPL Hooks Testnet. This is a parallel XRPL network - not the production one - where users can learn and experiment with deploying hooks without a risk to losing real money. The *Hooks Testnet* also offers its own *Faucet*, where users can get a *Hooks Testnet* account and "fake" money (fake XRP).

Backend. The Backend is an interface between the Frontend and the Remote Wasm Compiler. After the *Hook Smart Contract* is built, the user compiles the translated C code to *wasm*: though the push of a button on the Frontend, the C code is sent to the *Remote Wasm Compiler Server*, to be compiled to *wasm*. This process is managed by the *Backend*, which takes the C code from the Frontend and forwards it to the *Remote wasm compiler*. In turn, the *Remote Wasm Compiler Server* will return to the Backend the hook as a wasm file or the compilation error(s). The Backend will forward the received hook, or the error(s), to the Frontend which will display them on-screen to the user.

For security reasons, the rest of the process is handled on the Frontend: after receiving the hook as wasm code, the user signs the transaction and sends it to the Testnet for deployment. This must happen on the Frontend because the signing process involves private credentials which in a real-life scenario should preferably remain at all times on the user's machine. After deploying the signed transaction, the user will get back to the Frontend the result of the transaction: either *success*, or *failure*.

The multi-user architecture is presented in Figure 5. Users having installed on their local machines the free and open-source code consisting of the two *Frontend* and *Backend* Blockly2Hooks modules bundled together, will directly connect to the *remote wasm compiler* and to the *XRPL Hooks Testnet*, and will be instantly able to design and deploy Hooks.

The Blockly2Hooks platform is released as an open-source project on GitHub[3]. We have integrated some example Hooks for making it easier to get started and we are looking forward to adding more examples.

IV. Related Work

Marks [27] identifies different levels of abstraction for smart contract editors, from syntactic graphical editors such as based for example on Blocky, through flow-based graphical editors alike Unreal Engine *Blueprint*, and to *forms* to be filled-in for the simplest cases. It argues that because the necessary level of abstraction is dependent on the intended audience and use-case, until the users are there, the abstraction level can only be guessed. However, new or complex situations are likely to continue to be tackled by programmers.

An interesting study on the deployment of real-life contracts as smart contracts on the Ethereum blockchain has been undertaken in [40], where important challenges like the complexity of contract clauses and the user privacy are being exposed. The usage of visual domain-specific languages for smart contract creation on Ethereum Solidity is studied in [10], [41] and a concrete solution is proposed under the name *DasContract*.

Actually, most of the previous work on making smart contract development more accessible to the public has been focusing on visual editors for Solidity on the Ethereum blockchain, possibly because this is the most popular smart contracts platform: Latte [42] is such an example, which also provides feedback regarding the Gas cost incurred by the smart contract being built. The authors of [43] employ an interesting machine learning approach to build a visual programming environment for Ethereum smart contracts. A visual programming solution based on YAWL [44] is proposed by [19] to build smart contracts for Solidity, while working on a concrete use case from the construction industry. For a specific industry client, [45] designed and implemented a VPL-based

[3]https://github.com/wshbair/blockly2hooks

smart contract programming environment for a *legal purchase agreements* use case. Working on the same concrete case of Solidity and Ethereum, the authors of [46] identify a lack of smart contract descriptors (descriptors provide the information required to interact with a smart contract [46]), investigate the reusing of smart contracts, and propose then implement a design for smart contract descriptors, a descriptors registry, and a visual editor based on Google Blockly for creating composite smart contracts (smart contracts that call and interact with other smart contracts). The author of [18] identifies several requirements for widespread adoption of smart contracts in business and industry: ease of use, understandability, ease of testing, secure and error-free, scalable and affordable. End-user affordability is achieved mainly by replacing the programmers for smart contract creation [18]. Next, it investigates the use of declarative languages to enable beginners to easily and securely generate smart contracts, by working on a *"Will and Testament"* concrete use case. *FlowContract*[4] aims to be a flow-based smart contract editor for Ethereum Solidity, which however continues to expose lots of low-level programming elements to the user.

Blocks[5] [47] is an online smart contract editor for the *Internet Computer*[6]. It is a flow-oriented design which retains many syntactic elements, as most of the blocks, fields and variable names making up the visual design come straight from their text-programming counterparts. Another design is proposed in [48] for Hyperledger Fabric, however because of ties to a commercial project only limited work seems to be open-source and freely available.

Concerning Hooks, the XRPL Lab proposes an online, browser-based *Builder* [49] for the XRPL Hooks which puts together the hook documentation, hook examples, and the deployment to the Hooks Testnet. The creation of the smart contracts is carried over in the C language though, which ultimately makes Builder a tool for programmers. Same as *Builder*, Blockly2Hooks could also eliminate any need for the users to install code on their machines, i.e. place the functionalities still present on the user's machine on the remote server also, such that users just open their browser to access a remotely-served web interface from where they do everything.

To the best of our knowledge no other open-source project addresses visual smart contract development for contracts that are natively developed with advanced programming languages such as the modified C language used for the Hook smart contracts on the XRP Ledger. The environment proposed by Blockly2Hooks is general enough to accommodate the development of any kind of smart contract.

V. Conclusions and Future Work

For non-expert audience, developing smart contracts on Distributed Ledger Technology is currently a kind of mystery, with the field "reserved" to a narrow segment of seasoned professionals. In this work we investigated how interested audience with low programming skills can learn to develop smart contracts written even in complex languages such as the C language. We propose a classic, proven methodology for teaching programming which is shown to achieve good results: the Visual block-based icons and a Drag&Drop programming environment which is successfully used in schools to teach kids how to program. As a concrete case we used the Hooks smart contracts deployed on the XRP Ledger, and the visual programming environment we propose is built on Google's Blockly, a comprehensive but in the same time flexible, friendly and future-proof framework which offers possibilities for extending and improving the project.

The proposed design can help minimise the development time and costs, and increase the smart contract security by reducing the chances for smart contract bugs even for more tech-savvy users.

Blockly2Hooks was developed and tested and the results are promising to make learning smart contract development smoother.

Future work. The platform will be connected to Mainnet (the XRPL production network where real money are involved) as soon as the Hooks ammendment will be enabled there. As such, users will have the possibility to test their hooks on Testnet before deploying them on Mainnet. Additionally, we plan *focus testing* Blockly2Hooks, to gather more feedback from the community to make the system better and more useful. E.g., currently Blockly2Hooks is rather a syntactic graphical editor oriented towards learning smart contract development. Nevertheless, the platform is flexible enough to accommodate, in parallel, higher levels of abstraction: e.g., for the next version of Blockly2Hooks, we consider the implementation of templates, or macro-blocks, for some of the most sought after use cases. This will allow enriching the design to include a *flow-like* programming experience through interconnecting the macro blocks, treated by the user as black boxes.

[4]https://flowcontracts.com/docs, valid in February 2023

[5]https://blocks-editor.github.io/blocks/, valid in February 2023

[6]https://dfinity.org/, valid in February 2023

Acknowledgment

We thankfully acknowledge the support of the RIPPLE University Blockchain Research Initiative (UBRI) for our research.

References

[1] L. Trestioreanu, C. Nita-Rotaru, A. Malhotra, and R. State, "Spon: Enabling resilient inter-ledgers payments with an intrusion-tolerant overlay," in *2021 IEEE Conference on Communications and Network Security (CNS)*, 2021, pp. 92–100.

[2] The Interledger Foundation, "The interledger foundation," 2016, accessed: Feb. 2023. [Online]. Available: https://interledger.org/

[3] L. Ante, "The non-fungible token (nft) market and its relationship with bitcoin and ethereum," *FinTech*, vol. 1, no. 3, pp. 216–224, 2022.

[4] M. Nadini, L. Alessandretti, F. Di Giacinto, M. Martino, L. M. Aiello, and A. Baronchelli, "Mapping the nft revolution: market trends, trade networks, and visual features," *Scientific reports*, vol. 11, no. 1, p. 20902, 2021.

[5] L. Ante, "Non-fungible token (nft) markets on the ethereum blockchain: Temporal development, cointegration and interrelations," *Economics of Innovation and New Technology*, pp. 1–19, 2022.

[6] M. Kolvart, M. Poola, and A. Rull, "Smart contracts," *The Future of Law and etechnologies*, pp. 133–147, 2016.

[7] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE Transactions on Software Engineering*, vol. 47, no. 10, pp. 2084–2106, 2019.

[8] N. Szabo, "The idea of smart contracts," online, accessed: Feb. 2023. [Online]. Available: https://nakamotoinstitute.org/the-idea-of-smart-contracts/

[9] N. Szabo, "Smart contracts," online, accessed: Feb. 2023. [Online]. Available: https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html

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

[11] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, 1997.

[12] V. Buterin, "Ethereum whitepaper," 2014, accessed: Feb. 2023. [Online]. Available: https://ethereum.org/en/learn/

[13] V. Buterin, "Ethereum whitepaper," 2023, accessed: Feb. 2023. [Online]. Available: https://ethereum.org/en/whitepaper/

[14] XRP Ledger Foundation, "rippled," online, 02 2021, accessed: Jan. 2023. [Online]. Available: https://github.com/XRPLF/rippled/blob/develop/RELEASENOTES.md

[15] L. Mauri, S. Cimato, and E. Damiani, "A formal approach for the analysis of the xrp ledger consensus protocol," in *6th International Conference on Information Systems Security and Privacy, ICISSP*, 02 2020.

[16] Gavin Wood et al., "Ethereum: A secure decentralised generalised transaction ledger," 09 2017, accessed: Feb. 2023. [Online]. Available: https://archive.org/details/Ethereum_Yellow_Paper_201709

[17] B. Ray, D. Posnett, P. Devanbu, and V. Filkov, "A large-scale study of programming languages and code quality in github," *Commun. ACM*, vol. 60, no. 10, p. 91–100, sep 2017. [Online]. Available: https://doi.org/10.1145/3126905

[18] K. J. Purnell, "Towards declarative smart contracts," Ph.D. dissertation, Macquarie University, 2022.

[19] X. Ye and M. König, "From the graphical representation to the smart contract language: a use case in the construction industry," in *Proceedings of the 38th International Symposium on Automation and Robotics in Construction (ISARC)*, 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, and Z. Zhu, Eds. Dubai, UAE: International Association for Automation and Robotics in Construction (IAARC), November 2021, pp. 272–279.

[20] Webflow, "The site you want — without the dev time," 2023, accessed: Feb. 2023. [Online]. Available: https://webflow.com/about

[21] Bubble.io, "The best way to build marketplaces without code," 2023, accessed: Feb. 2023. [Online]. Available: https://bubble.io/

[22] I. N. Oteyo, A. L. S. Pupo, J. Zaman, S. Kimani, W. De Meuter, and E. G. Boix, "Building smart agriculture applications using low-code tools: The case for discopar," in *2021 IEEE AFRICON*, 2021, pp. 1–6.

[23] B. Jost, M. Ketterl, R. Budde, and T. Leimbach, "Graphical programming environments for educational robots: Open roberta - yet another one?" in *2014 IEEE International Symposium on Multimedia*, 2014, pp. 381–386.

[24] M. A. Kuhail, S. Farooq, R. Hammad, and M. Bahja, "Characterizing visual programming approaches for end-user developers: A systematic review," *IEEE Access*, vol. 9, pp. 14 181–14 202, 2021.

[25] S. Bragg and C. Driskill, "Diagrammatic-graphical programming languages and dod-std-2167a," in *Proceedings of AUTOTESTCON '94*, 1994, pp. 211–220.

[26] Wikipedia, "Visual programming language," 2023, accessed: Feb. 2023. [Online]. Available: https://en.wikipedia.org/wiki/Visual_programming_language#cite_note-2

[27] E. Marks, "The case for graphical smart contract editors," 04 2018, accessed: Feb. 2023. [Online]. Available: https://medium.com/pennblockchain/the-case-for-graphical-smart-contract-editors-8e721cdcde93

[28] V. Dhillon, D. Metcalf, and M. Hooper, *The DAO Hacked*. Berkeley, CA: Apress, 2017, pp. 67–78. [Online]. Available: https://doi.org/10.1007/978-1-4842-3081-7_6

[29] Y. Ikeda, Y. Ohki, Z. Marquardt, Y. Kimura, S. Omura, and E. Yoshikawa, "First demonstration experiment for energy trading system edison-x using the xrp ledger," *arXiv preprint arXiv:2212.02044*, 2022.

[30] L. Trestioreanu, W. M. Shbair, F. S. de Cristo, and R. State, "XRP-NDN Overlay: Improving the communication efficiency of consensus-validation based blockchains with an NDN Overlay," *arXiv preprint arXiv:2301.10209*, 2023.

[31] I. Amores-Sesar, C. Cachin, and J. Mićić, "Security analysis of ripple consensus," 2020. [Online]. Available: https://arxiv.org/abs/2011.14816

[32] XRPL Labs, "A smart contract proposal for the xrp ledger," 2022, accessed: Feb. 2023. [Online]. Available: https://hooks.xrpl.org/

[33] XRPL Labs, "XRPL Hooks amendment," 2023, accessed: Feb. 2023. [Online]. Available: https://hooks-testnet-v2.xrpl-labs.com/

[34] XRPL Labs, "Hooks technology preview quickstart," online, accessed: Feb. 2023. [Online]. Available: https://github.com/XRPL-Labs/xrpld-hooks/tree/hooks-ssvm/hook-api-examples

[35] MIT, "Scratch," online, accessed: Feb. 2023. [Online]. Available: https://scratch.mit.edu/

[36] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, pp. 60–67, 11 2009.

[37] A. Bau, "Droplet: Blocks and text together," 2023, accessed: Feb. 2023. [Online]. Available: https://droplet-editor.github.io/

[38] M. S. Lab, "Taleblazer," 2023, accessed: Feb. 2023. [Online]. Available: https://education.mit.edu/project/taleblazer/

[39] Google, "Blockly," online, accessed: Feb. 2023. [Online]. Available: https://developers.google.com/blockly

[40] W. Egbertsen, G. Hardeman, M. van den Hoven, G. van der Kolk, and A. van Rijsewijk, "Replacing paper contracts with ethereum smart contracts," 2016, accessed: Feb. 2023. [Online]. Available: https://allquantor.at/blockchainbib/pdf/egbertsen2016replacing.pdf

[41] M. Skotnica, J. Klicpera, and R. Pergl, "Towards model-driven smart contract systems - code generation and improving expressivity of smart contract modeling," in *Proceedings of the 20th CIAO! Doctoral Consortium, and Enterprise Engineering Working Conference Forum 2020*, 03 2021. [Online]. Available: https://ceur-ws.org/Vol-2825/

[42] S. Tan, S. S Bhowmick, H. E. Chua, and X. Xiao, "Latte: Visual construction of smart contracts," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2713–2716. [Online]. Available: https://doi.org/10.1145/3318464.3384687

[43] D. Mao, F. Wang, Y. Wang, and Z. Hao, "Visual and user-defined smart contract designing system based on automatic coding," *IEEE Access*, vol. 7, pp. 73 131–73 143, 2019.

[44] W. M. P. van der Aalst, L. Aldred, M. Dumas, and A. H. M. ter Hofstede, "Design and implementation of the yawl system," in *Advanced Information Systems Engineering*, A. Persson and J. Stirna, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 142–159.

[45] T. Weingaertner, R. Rao, J. Ettlin, P. Suter, and 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.

[46] L. Guida and 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.

[47] R. Vandersmith, "How we created blocks: an online drag-and-drop smart contract editor," 01 2022, accessed: Feb. 2023. [Online]. Available: https://levelup.gitconnected.com/how-we-created-blocks-an-online-drag-and-drop-smart-contract-editor-fe23eff4d933

[48] M. M. Merlec, Y. K. Lee, and 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.

[49] XRPL Labs, "XRPL Hooks builder," 2023, accessed: Feb. 2023. [Online]. Available: https://hooks-builder.xrpl.org/develop